Verified Construction of Fair Voting Rules

Michael Kirsten

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
kirsten@kit.edu

July 5, 2021

Abstract

Voting rules aggregate multiple individual preferences in order to make a collective decision. Commonly, these mechanisms are expected to respect a multitude of different notions of fairness and reliability, which must be carefully balanced to avoid inconsistencies.

This article contains a formalisation of a framework for the construction of such fair voting rules using composable modules [1, 2]. The framework is a formal and systematic approach for the flexible and verified construction of voting rules from individual composable modules to respect such social-choice properties by construction. Formal composition rules guarantee resulting social-choice properties from properties of the individual components which are of generic nature to be reused for various voting rules. We provide proofs for a selected set of structures and composition rules. The approach can be readily extended in order to support more voting rules, e.g., from the literature by extending the sets of modules and composition rules.

Contents

Chapter 1

Social-Choice Types

1.1 Preference Relation

theory *Preference-Relation*
imports *Main*
begin

The very core of the composable modules voting framework: types and functions, derivations, lemmata, operations on preference relations, etc.

1.1.1 Definition

type-synonym $'a$ *Preference-Relation* $=$ $'a$ *rel*

fun *is-less-preferred-than* ::
$'a \Rightarrow 'a$ *Preference-Relation* $\Rightarrow 'a \Rightarrow bool$ (- $\preceq$- - [*50*, *1000*, *51*] *50*) **where**
$x \preceq_r y = ((x, y) \in r)$

lemma *lin-imp-antisym*:
assumes *linear-order-on A r*
shows *antisym r*
using *assms linear-order-on-def partial-order-on-def*
by *auto*

lemma *lin-imp-trans*:
assumes *linear-order-on A r*
shows *trans r*
using *assms order-on-defs*
by *blast*

1.1.2 Ranking

fun *rank* :: $'a$ *Preference-Relation* $\Rightarrow 'a \Rightarrow nat$ **where**
rank r x $=$ *card* (*above r x*)

lemma *rank-gt-zero*:
assumes
refl: $x \preceq_r x$ **and**
fin: *finite r*
shows $rank\ r\ x \geq 1$
proof −
have $x \in \{y \in Field\ r.\ (x, y) \in r\}$
using *FieldI2 refl*
by *fastforce*
hence $\{y \in Field\ r.\ (x, y) \in r\} \neq \{\}$
by *blast*
hence $card\ \{y \in Field\ r.\ (x, y) \in r\} \neq 0$
by (*simp add*: *fin finite-Field*)
moreover have $card\{y \in Field\ r.\ (x, y) \in r\} \geq 0$
using *fin*
by *auto*
ultimately show *?thesis*
using *Collect-cong FieldI2 above-def*
less-one not-le-imp-less rank.elims
by (*metis* (*no-types*, *lifting*))
qed

1.1.3 Limited Preference

definition *limited* :: $'a\ set \Rightarrow 'a\ Preference\text{-}Relation \Rightarrow bool$ **where**
$limited\ A\ r \equiv r \subseteq A \times A$

lemma *limitedI*:
$(\bigwedge x\ y.\ [\![\ x \preceq_r y\ ]\!] \Longrightarrow x \in A \land y \in A) \Longrightarrow limited\ A\ r$
unfolding *limited-def*
by *auto*

lemma *limited-dest*:
$(\bigwedge x\ y.\ [\![\ x \preceq_r y;\ limited\ A\ r\ ]\!] \Longrightarrow x \in A \land y \in A)$
unfolding *limited-def*
by *auto*

fun *limit* :: $'a\ set \Rightarrow 'a\ Preference\text{-}Relation \Rightarrow 'a\ Preference\text{-}Relation$ **where**
$limit\ A\ r = \{(a, b) \in r.\ a \in A \land b \in A\}$

definition *connex* :: $'a\ set \Rightarrow 'a\ Preference\text{-}Relation \Rightarrow bool$ **where**
$connex\ A\ r \equiv limited\ A\ r \land (\forall x \in A.\ \forall y \in A.\ x \preceq_r y \lor y \preceq_r x)$

lemma *connex-imp-refl*:
assumes *connex A r*
shows *refl-on A r*
proof
show $r \subseteq A \times A$
using *assms connex-def limited-def*

by *metis*
next
fix
$x :: 'a$
assume
x-in-A: $x \in A$
have $x \preceq_r x$
using *assms connex-def x-in-A*
by *metis*
thus $(x, x) \in r$
by *simp*
qed

lemma *lin-ord-imp-connex*:
assumes *linear-order-on* A r
shows *connex* A r
unfolding *connex-def limited-def*
proof (*safe*)
fix
$a :: 'a$ **and**
$b :: 'a$
assume
asm1: $(a, b) \in r$
show $a \in A$
using *asm1 assms partial-order-onD(1) order-on-defs(3) refl-on-domain*
by *metis*
next
fix
$a :: 'a$ **and**
$b :: 'a$
assume
asm1: $(a, b) \in r$
show $b \in A$
using *asm1 assms partial-order-onD(1) order-on-defs(3) refl-on-domain*
by *metis*
next
fix
$x :: 'a$ **and**
$y :: 'a$
assume
asm1: $x \in A$ **and**
asm2: $y \in A$ **and**
asm3: $\neg\, y \preceq_r x$
have $(y, x) \notin r$
using *asm3*
by *simp*
hence $(x, y) \in r$

```
    using asm1 asm2 assms partial-order-onD(1)
          linear-order-on-def refl-onD total-on-def
    by metis
  thus x ≼r y
    by simp
qed

lemma connex-antsym-and-trans-imp-lin-ord:
  assumes
    connex-r: connex A r and
    antisym-r: antisym r and
    trans-r: trans r
  shows linear-order-on A r
  unfolding connex-def linear-order-on-def partial-order-on-def
            preorder-on-def refl-on-def total-on-def
proof (safe)
  fix
    a :: 'a and
    b :: 'a
  assume
    asm1: (a, b) ∈ r
  show a ∈ A
    using asm1 connex-r refl-on-domain connex-imp-refl
    by metis
next
  fix
    a :: 'a and
    b :: 'a
  assume
    asm1: (a, b) ∈ r
  show b ∈ A
    using asm1 connex-r refl-on-domain connex-imp-refl
    by metis
next
  fix
    x :: 'a
  assume
    asm1: x ∈ A
  show (x, x) ∈ r
    using asm1 connex-r connex-imp-refl refl-onD
  by metis
next
  show trans r
    using trans-r
    by simp
next
  show antisym r
    using antisym-r
    by simp
```

next
  fix
    $x :: {}'a$ **and**
    $y :: {}'a$
  assume
    asm1: $x \in A$ **and**
    asm2: $y \in A$ **and**
    asm3: $x \neq y$ **and**
    asm4: $(y, x) \notin r$
  have $x \preceq_r y \vee y \preceq_r x$
    using *asm1 asm2 connex-r connex-def*
    by *metis*
  hence $(x, y) \in r \vee (y, x) \in r$
    by *simp*
  thus $(x, y) \in r$
    using *asm4*
    by *metis*
qed

lemma *limit-to-limits*: *limited* A (*limit* A r)
  unfolding *limited-def*
  by *auto*

lemma *limit-presv-connex*:
  assumes
    connex: *connex* S r **and**
    subset: $A \subseteq S$
  shows *connex* A (*limit* A r)
  unfolding *connex-def limited-def*
proof (*simp*, *safe*)
  let $?s = \{(a, b).\ (a, b) \in r \wedge a \in A \wedge b \in A\}$
  fix
    $x :: {}'a$ **and**
    $y :: {}'a$ **and**
    $a :: {}'a$ **and**
    $b :: {}'a$
  assume
    asm1: $x \in A$ **and**
    asm2: $y \in A$ **and**
    asm3: $(y, x) \notin r$
  have $y \preceq_r x \vee x \preceq_r y$
    using *asm1 asm2 connex connex-def in-mono subset*
    by *metis*
  hence
    $x \preceq_{?s} y \vee y \preceq_{?s} x$
    using *asm1 asm2*
    by *auto*
  hence $x \preceq_{?s} y$
    using *asm3*

```
    by simp
  thus (x, y) ∈ r
    by simp
qed

lemma limit-presv-antisym:
  assumes
    antisymmetric: antisym r and
    subset: A ⊆ S
  shows antisym (limit A r)
  using antisym-def antisymmetric
  by auto

lemma limit-presv-trans:
  assumes
    transitive: trans r and
    subset:     A ⊆ S
  shows trans (limit A r)
  unfolding trans-def
proof (simp, safe)
  fix
    x :: 'a and
    y :: 'a and
    z :: 'a
  assume
    asm1: (x, y) ∈ r and
    asm2: x ∈ A and
    asm3: y ∈ A and
    asm4: (y, z) ∈ r and
    asm5: z ∈ A
  show  (x, z) ∈ r
    using asm1 asm4 transE transitive
    by metis
qed

lemma limit-presv-lin-ord:
  assumes
    linear-order-on S r and
      A ⊆ S
    shows linear-order-on A (limit A r)
  using assms connex-antsym-and-trans-imp-lin-ord
          limit-presv-antisym limit-presv-connex
          limit-presv-trans lin-ord-imp-connex
          order-on-defs(1) order-on-defs(2)
          order-on-defs(3)
  by metis

lemma limit-presv-prefs1:
  assumes
```

x-less-y: $x \preceq_r y$ **and**
x-in-A: $x \in A$ **and**
y-in-A: $y \in A$
shows *let* s = *limit A r in* $x \preceq_s y$
using *x-in-A x-less-y y-in-A*
by *simp*

lemma *limit-presv-prefs2*:
assumes *x-less-y*: $(x, y) \in$ *limit A r*
shows $x \preceq_r y$
using *mem-Collect-eq x-less-y*
by *auto*

lemma *limit-trans*:
assumes
$B \subseteq A$ **and**
$C \subseteq B$ **and**
linear-order-on A r
shows *limit C r* = *limit C* (*limit B r*)
using *assms*
by *auto*

lemma *lin-ord-not-empty*:
assumes $r \neq \{\}$
shows $\neg$ *linear-order-on* $\{\}$ *r*
using *assms connex-imp-refl lin-ord-imp-connex*
refl-on-domain subrelI
by *fastforce*

lemma *lin-ord-singleton*:
$\forall r$. *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
proof
fix *r* :: *'a Preference-Relation*
show *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
proof
assume *asm*: *linear-order-on* $\{a\}$ *r*
hence $a \preceq_r a$
using *connex-def lin-ord-imp-connex singletonI*
by *metis*
moreover have $\forall (x, y) \in r.\ x = a \wedge y = a$
using *asm connex-imp-refl lin-ord-imp-connex*
refl-on-domain split-beta
by *fastforce*
ultimately show $r = \{(a, a)\}$
by *auto*
qed
qed

1.1.4 Auxiliary Lemmata

lemma *above-trans*:
  assumes
    trans r **and**
    $(a, b) \in r$
  shows *above* r $b \subseteq$ *above* r a
  using *Collect-mono above-def assms transE*
  by *metis*

lemma *above-refl*:
  assumes
    refl-on A r **and**
    $a \in A$
  shows $a \in$ *above* r a
  using *above-def assms refl-onD*
  by *fastforce*

lemma *above-subset-geq-one*:
  assumes
    linear-order-on A r $\wedge$ *linear-order-on* A s **and**
    above r $a \subseteq$ *above* s a **and**
    above s $a = \{a\}$
  shows *above* r $a = \{a\}$
  using *above-def assms connex-imp-refl above-refl insert-absorb*
      lin-ord-imp-connex mem-Collect-eq refl-on-domain
      singletonI subset-singletonD
  by *metis*

lemma *above-connex*:
  assumes
    connex A r **and**
    $a \in A$
  shows $a \in$ *above* r a
  using *assms connex-imp-refl above-refl*
  by *metis*

lemma *pref-imp-in-above*: $a \preceq_r b \longleftrightarrow b \in$ *above* r a
  by (*simp add*: *above-def*)

lemma *limit-presv-above*:
  assumes
    $b \in$ *above* r a **and**

    $a \in B \wedge b \in B$
  shows $b \in$ *above* (*limit* B r) a
  using *pref-imp-in-above assms limit-presv-prefs1*
  by *metis*

lemma *limit-presv-above2*:

```
  assumes
    b ∈ above (limit B r) a and
    linear-order-on A r and
    B ⊆ A and
    a ∈ B and
    b ∈ B
  shows b ∈ above r a
  unfolding above-def
  using above-def assms(1) limit-presv-prefs2
        mem-Collect-eq pref-imp-in-above
  by metis

lemma above-one:
  assumes
    linear-order-on A r and
    finite A ∧ A ≠ {}
  shows ∃ a∈A. above r a = {a} ∧ (∀ x∈A. above r x = {x} ⟶ x = a)
proof −
  obtain n::nat where n: n+1 = card A
    using Suc-eq-plus1 antisym-conv2 assms(2) card-eq-0-iff
          gr0-implies-Suc le0
    by metis
  have
    (linear-order-on A r ∧ finite A ∧ A ≠ {} ∧ n+1 = card A)
          ⟶ (∃ a. a∈A ∧ above r a = {a})
  proof (induction n arbitrary: A r)
    case 0
    show ?case
    proof
      assume asm: linear-order-on A r ∧ finite A ∧ A ≠ {} ∧ 0+1 = card A
      then obtain a where {a} = A
        using card-1-singletonE add.left-neutral
        by metis
      hence a ∈ A ∧ above r a = {a}
        using above-def asm connex-imp-refl above-refl
              lin-ord-imp-connex refl-on-domain
        by fastforce
      thus ∃ a. a∈A ∧ above r a = {a}
        by auto
    qed
  next
    case (Suc n)
    show ?case
    proof
      assume asm:
        linear-order-on A r ∧ finite A ∧ A ≠ {} ∧ Suc n+1 = card A
      then obtain B where B: card B = n+1 ∧ B ⊆ A
        using Suc-inject add-Suc card.insert-remove finite.cases
              insert-Diff-single subset-insertI
```

by (*metis* (*mono-tags*, *lifting*))
then obtain a **where** a: $\{a\} = A - B$
using *Suc-eq-plus1 add-diff-cancel-left′ asm card-1-singletonE card-Diff-subset finite-subset*
by *metis*
have $\exists\, b{\in}B.\ above\ (limit\ B\ r)\ b = \{b\}$
using *B One-nat-def Suc.IH add-diff-cancel-left′ asm card-eq-0-iff diff-le-self finite-subset leD lessI limit-presv-lin-ord*
by *metis*
then obtain b **where** b: $above\ (limit\ B\ r)\ b = \{b\}$
by *blast*
show $\exists\, a.\ a \in A \wedge above\ r\ a = \{a\}$
proof *cases*
assume
asm1: $a \preceq_r b$
have *f1*:
$\forall\, A\ r\ a\ aa.$
$\neg\ refl\text{-}on\ A\ r \vee (a{::}'a,\ aa) \notin r \vee a \in A \wedge aa \in A$
using *refl-on-domain*
by *metis*
have *f2*:
$\forall\, A\ r.\ \neg\ connex\ (A{::}'a\ set)\ r \vee refl\text{-}on\ A\ r$
using *connex-imp-refl*
by *metis*
have *f3*:
$\forall\, A\ r.\ \neg\ linear\text{-}order\text{-}on\ (A{::}'a\ set)\ r \vee connex\ A\ r$
by (*simp add*: *lin-ord-imp-connex*)
hence $refl\text{-}on\ A\ r$
using *f2 asm*
by *metis*
hence $a \in A \wedge b \in A$
using *f1 asm1*
by *simp*
hence *f4*:
$\forall\, a.\ a \notin A \vee b = a \vee (b,\ a) \in r \vee (a,\ b) \in r$
using *asm order-on-defs*(*3*) *total-on-def*
by *metis*
have *f5*:
$(b,\ b) \in limit\ B\ r$
using *above-def b mem-Collect-eq singletonI*
by *metis*
have *f6*:
$\forall\, a\ A\ Aa.\ (a{::}'a) \notin A - Aa \vee a \in A \wedge a \notin Aa$
by *simp*
have *ff1*:
$\{a.\ (b,\ a) \in limit\ B\ r\} = \{b\}$
using *above-def b*
by (*metis* (*no-types*))

have *ff2*:
  $(b,\ b) \in \{(aa,\ a).\ (aa,\ a) \in r \wedge aa \in B \wedge a \in B\}$
  using *f5*
  by *simp*
moreover have *b-wins-B*:
  $\forall\, x \in B.\ b \in$ *above r x*
  using *B above-def f4 ff1 ff2 CollectI*
    Product-Type.Collect-case-prodD
  by *fastforce*
moreover have $b \in$ *above r a*
  using *asm1 pref-imp-in-above*
  by *metis*
ultimately have *b-wins*:
  $\forall\, x \in A.\ b \in$ *above r x*
  using *Diff-iff a empty-iff insert-iff*
  by (*metis* (*no-types*))
hence $\forall\, x \in A.\ x \in$ *above r b* $\longrightarrow x = b$
  using *CollectD above-def antisym-def asm lin-imp-antisym*
  by *metis*
hence $\forall\, x \in A.\ x \in$ *above r b* $\longleftrightarrow x = b$
  using *b-wins*
  by *blast*
moreover have *above-b-in-A*: *above r b* $\subseteq A$
  using *above-def asm connex-imp-refl lin-ord-imp-connex*
    mem-Collect-eq refl-on-domain subsetI
  by *metis*
ultimately have *above r b* $= \{b\}$
  using *above-def b*
  by *fastforce*
thus *?thesis*
  using *above-b-in-A*
  by *blast*
next
assume $\neg a \preceq_r b$
hence *b-smaller-a*: $b \preceq_r a$
  using *B DiffE a asm b limit-to-limits connex-def*
    limited-dest singletonI subset-iff
    lin-ord-imp-connex pref-imp-in-above
  by *metis*
hence *b-smaller-a-0*: $(b,\ a) \in r$
  by *simp*
have *g1*:
  $\forall\, A\ r\ Aa.$
    $\neg$ *linear-order-on* $(A::'a\ set)$ $r \vee$
     $\neg$ $Aa \subseteq A \vee$
     linear-order-on Aa (*limit Aa r*)
  using *limit-presv-lin-ord*
  by *metis*
have

$\{a.\ (b,\ a) \in \mathit{limit}\ B\ r\} = \{b\}$
using *above-def b*
by *metis*
hence *g2*: $b \in B$
by *auto*
have *g3*:
$\mathit{partial\text{-}order\text{-}on}\ B\ (\mathit{limit}\ B\ r) \wedge \mathit{total\text{-}on}\ B\ (\mathit{limit}\ B\ r)$
using *g1 B asm order-on-defs*(*3*)
by *metis*
have
$\forall A\ r.$
$\mathit{total\text{-}on}\ A\ r = (\forall a.\ (a{::}'a) \notin A\ \vee$
$(\forall aa.\ (aa \notin A \vee a = aa) \vee (a,\ aa) \in r \vee (aa,\ a) \in r))$
using *total-on-def*
by *metis*
hence
$\forall a.\ a \notin B\ \vee$
$(\forall aa.\ aa \notin B \vee a = aa\ \vee$
$(a,\ aa) \in \mathit{limit}\ B\ r \vee (aa,\ a) \in \mathit{limit}\ B\ r)$
using *g3*
by *simp*
have $\forall x \in B.\ b \in \mathit{above}\ r\ x$
using *limit-presv-above2 B pref-imp-in-above asm b above-def*
limit-presv-lin-ord order-on-defs(*3*) *singletonD*
singletonI total-on-def mem-Collect-eq g2
by (*smt* (*verit*, *ccfv-threshold*))
hence *b-wins2*:
$\forall x \in B.\ x \preceq_r b$
by (*simp add*: *above-def*)
hence *b-wins2-0*:
$\forall x \in B.\ (x,\ b) \in r$
by *simp*
have *trans r*
using *asm lin-imp-trans*
by *metis*
hence $\forall x \in B.\ (x,\ a) \in r$
using *transE b-smaller-a-0 b-wins2-0*
by *metis*
hence $\forall x \in B.\ x \preceq_r a$
by *simp*
hence *nothing-above-a*: $\forall x \in A.\ x \preceq_r a$
using *a asm lin-ord-imp-connex above-connex Diff-iff*
empty-iff insert-iff pref-imp-in-above
by *metis*
have $\forall x \in A.\ x \in \mathit{above}\ r\ a \longleftrightarrow x = a$
using *antisym-def asm lin-imp-antisym*
nothing-above-a pref-imp-in-above
CollectD above-def
by *metis*

moreover have *above-a-in-A*: *above r a* $\subseteq$ *A*
using *above-def asm connex-imp-refl lin-ord-imp-connex*
mem-Collect-eq refl-on-domain
by *fastforce*
ultimately have *above r a* = $\{a\}$
using *above-def a*
by *auto*
thus *?thesis*
using *above-a-in-A*
by *blast*
qed
qed
qed
hence $\exists$ *a*. *a*$\in$*A* $\wedge$ *above r a* = $\{a\}$
using *assms n*
by *blast*
thus *?thesis*
using *Diff-eq-empty-iff above-trans assms*(*1*) *empty-Diff insertE*
insert-Diff-if insert-absorb insert-not-empty order-on-defs(*1*)
order-on-defs(*2*) *order-on-defs*(*3*) *total-on-def*
by (*smt* (*verit*, *ccfv-SIG*))
qed

lemma *above-one2*:
assumes
lin-ord: *linear-order-on A r* **and**
fin-not-emp: *finite A* $\wedge$ *A* $\neq$ $\{\}$ **and**
above1: *above r a* = $\{a\}$ $\wedge$ *above r b* = $\{b\}$
shows *a* = *b*
proof −
have $a \preceq_r a \wedge b \preceq_r b$
using *above1 singletonI pref-imp-in-above*
by *metis*
also have
$\exists$ *a*$\in$*A*. *above r a* = $\{a\}$ $\wedge$
($\forall$ *x*$\in$*A*. *above r x* = $\{x\}$ $\longrightarrow$ *x* = *a*)
using *lin-ord fin-not-emp*
by (*simp add*: *above-one*)
moreover have *connex A r*
using *lin-ord*
by (*simp add*: *lin-ord-imp-connex*)
ultimately show *a* = *b*
using *above1 connex-def limited-dest*
by *metis*
qed

lemma *above-presv-limit*:
assumes *linear-order r*
shows *above* (*limit A r*) *x* $\subseteq$ *A*

unfolding *above-def*
by *auto*

1.1.5 Lifting Property

definition *equiv-rel-except-a* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$
$'a$ *Preference-Relation* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
equiv-rel-except-a $A\ r\ s\ a \equiv$
linear-order-on $A\ r \land$ *linear-order-on* $A\ s \land a \in A \land$
$(\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y)$

definition *lifted* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$
$'a$ *Preference-Relation* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
lifted $A\ r\ s\ a \equiv$
equiv-rel-except-a $A\ r\ s\ a \land (\exists x \in A - \{a\}.\ a \preceq_r x \land x \preceq_s a)$

lemma *trivial-equiv-rel*:
assumes *order*: *linear-order-on* $A\ p$
shows $\forall a \in A.$ *equiv-rel-except-a* $A\ p\ p\ a$
by (*simp add*: *equiv-rel-except-a-def order*)

lemma *lifted-imp-equiv-rel-except-a*:
assumes *lifted*: *lifted* $A\ r\ s\ a$
shows *equiv-rel-except-a* $A\ r\ s\ a$
proof –
from *lifted* **have**
linear-order-on $A\ r \land$ *linear-order-on* $A\ s \land a \in A \land$
$(\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y)$
by (*simp add*: *lifted-def equiv-rel-except-a-def*)
thus *?thesis*
by (*simp add*: *equiv-rel-except-a-def*)
qed

lemma *lifted-mono*:
assumes *lifted*: *lifted* $A\ r\ s\ a$
shows $\forall x \in A - \{a\}.\ \neg(x \preceq_r a \land a \preceq_s x)$
proof (*safe*)
fix
$x :: 'a$
assume
x-in-A: $x \in A$ **and**
x-exist: $x \notin \{\}$ **and**
x-neq-a: $x \neq a$ **and**
x-pref-a: $x \preceq_r a$ **and**
a-pref-x: $a \preceq_s x$
from *x-pref-a*
have *x-pref-a-0*: $(x, a) \in r$
by *simp*
from *a-pref-x*

have *a-pref-x-0*: $(a, x) \in s$
  by *simp*
have *antisym r*
  using *equiv-rel-except-a-def lifted*
      lifted-imp-equiv-rel-except-a
      lin-imp-antisym
  by *metis*
hence *antisym-r*:
  $(\forall x\ y.\ (x, y) \in r \longrightarrow (y, x) \in r \longrightarrow x = y)$
  using *antisym-def*
  by *metis*
hence *imp-x-eq-a-0*:
  $[\![(x, a) \in r; (a, x) \in r]\!] \Longrightarrow x = a$
  by *simp*
have *lift-ex*: $\exists x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
  using *lifted lifted-def*
  by *metis*
from *lift-ex* **obtain** $y :: 'a$ **where**
  f1: $y \in A - \{a\} \wedge a \preceq_r y \wedge y \preceq_s a$
  by *metis*
hence *f1-0*:
  $y \in A - \{a\} \wedge (a, y) \in r \wedge (y, a) \in s$
  by *simp*
have *f2*:
  equiv-rel-except-a A r s a
  using *lifted lifted-def*
  by *metis*
hence *f2-0*:
  $\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
  using *equiv-rel-except-a-def*
  by *metis*
hence *f2-1*:
  $\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ (x, y) \in r \longleftrightarrow (x, y) \in s$
  by *simp*
have *trans*: $\forall x\ y\ z.\ (x, y) \in r \longrightarrow (y, z) \in r \longrightarrow (x, z) \in r$
  using *f2 equiv-rel-except-a-def linear-order-on-def*
      partial-order-on-def preorder-on-def trans-def
  by *metis*
have *x-pref-y-0*: $(x, y) \in s$
  using *equiv-rel-except-a-def f1-0 f2 f2-1 insertE*
      insert-Diff x-in-A x-neq-a x-pref-a-0 trans
  by *metis*
have *a-pref-y-0*: $(a, y) \in s$
  using *a-pref-x-0 imp-x-eq-a-0 x-neq-a x-pref-a-0*
      equiv-rel-except-a-def f2 lin-imp-trans
      transE x-pref-y-0
  by *metis*
show *False*
  using *a-pref-y-0 antisymD equiv-rel-except-a-def*

DiffD2 f1-0 f2 lin-imp-antisym singletonI
by *metis*
qed

lemma *lifted-mono2*:
assumes
lifted: *lifted* $A\ r\ s\ a$ **and**
x-pref-a: $x \preceq_r a$
shows $x \preceq_s a$
proof (*simp*)
have *x-pref-a-0*: $(x, a) \in r$
using *x-pref-a*
by *simp*
have *x-in-A*: $x \in A$
using *connex-imp-refl equiv-rel-except-a-def*
lifted lifted-def lin-ord-imp-connex
refl-on-domain x-pref-a-0
by *metis*
have $\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
using *lifted lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*:
$\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ (x, y) \in r \longleftrightarrow (x, y) \in s$
by *simp*
have $\exists x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
using *lifted lifted-def*
by *metis*
hence *ex-lifted*:
$\exists x \in A - \{a\}.\ (a, x) \in r \wedge (x, a) \in s$
by *simp*
show $(x, a) \in s$
proof (*cases* $x = a$)
case *True*
thus *?thesis*
using *connex-imp-refl equiv-rel-except-a-def refl-onD*
lifted lifted-def lin-ord-imp-connex
by *metis*
next
case *False*
thus *?thesis*
using *equiv-rel-except-a-def insertE insert-Diff*
lifted lifted-imp-equiv-rel-except-a x-in-A
x-pref-a-0 ex-lifted lin-imp-trans rest-eq
trans-def
by *metis*
qed
qed

lemma *lifted-above*:

assumes *lifted A r s a*
shows *above s a* $\subseteq$ *above r a*
unfolding *above-def*
proof (*safe*)
fix
$x :: 'a$
assume
a-pref-x: $(a, x) \in s$
have $\exists x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
using *assms lifted-def*
by *metis*
hence *lifted-r*:
$\exists x \in A - \{a\}.\ (a, x) \in r \wedge (x, a) \in s$
by *simp*
have $\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
using *assms lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*:
$\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ (x, y) \in r \longleftrightarrow (x, y) \in s$
by *simp*
have *trans-r*:
$\forall x\ y\ z.\ (x, y) \in r \longrightarrow (y, z) \in r \longrightarrow (x, z) \in r$
using *trans-def lifted-def lin-imp-trans*
equiv-rel-except-a-def assms
by *metis*
have *trans-s*:
$\forall x\ y\ z.\ (x, y) \in s \longrightarrow (y, z) \in s \longrightarrow (x, z) \in s$
using *trans-def lifted-def lin-imp-trans*
equiv-rel-except-a-def assms
by *metis*
have *refl-r*:
$(a, a) \in r$
using *assms connex-imp-refl equiv-rel-except-a-def*
lifted-def lin-ord-imp-connex refl-onD
by *metis*
have *x-in-A*: $x \in A$
using *a-pref-x assms connex-imp-refl equiv-rel-except-a-def*
lifted-def lin-ord-imp-connex refl-onD2
by *metis*
show $(a, x) \in r$
using *Diff-iff a-pref-x lifted-r rest-eq singletonD*
trans-r trans-s x-in-A refl-r
by (*metis* (*full-types*))
qed

lemma *lifted-above2*:
assumes
lifted A r s a **and**
$x \in A - \{a\}$

shows *above r x* $\subseteq$ *above s x* $\cup$ $\{a\}$
proof (*safe*, *simp*)
fix y :: $'a$
assume
y-in-above-r: $y \in$ *above r x* **and**
y-not-in-above-s: $y \notin$ *above s x*
have $\forall z \in A-\{a\}.\ x \preceq_r z \longleftrightarrow x \preceq_s z$
using *assms lifted-def equiv-rel-except-a-def*
by *metis*
hence $\forall z \in A-\{a\}.\ (x, z) \in r \longleftrightarrow (x, z) \in s$
by *simp*
hence $\forall z \in A-\{a\}.\ z \in$ *above r x* $\longleftrightarrow z \in$ *above s x*
by (*simp add*: *above-def*)
hence $y \in$ *above r x* $\longleftrightarrow y \in$ *above s x*
using *y-not-in-above-s assms*(*1*) *connex-def*
equiv-rel-except-a-def lifted-def lifted-mono2
limited-dest lin-ord-imp-connex member-remove
pref-imp-in-above remove-def
by *metis*
thus $y = a$
using *y-in-above-r y-not-in-above-s*
by *simp*
qed

lemma *limit-lifted-imp-eq-or-lifted*:
assumes
lifted: *lifted S r s a* **and**
subset: $A \subseteq S$
shows
limit A r = *limit A s* $\vee$
lifted A (*limit A r*) (*limit A s*) *a*
proof −
from *lifted* **have**
$\forall x \in S - \{a\}.\ \forall y \in S - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
by (*simp add*: *lifted-def equiv-rel-except-a-def*)
with *subset* **have** *temp*:
$\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
by *auto*
hence *eql-rs*:
$\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.$
$(x, y) \in$ (*limit A r*) $\longleftrightarrow (x, y) \in$ (*limit A s*)
using *DiffD1 limit-presv-prefs1 limit-presv-prefs2*
by *auto*
show *?thesis*
proof *cases*
assume *a1*: $a \in A$
thus *?thesis*
proof *cases*

assume *a1-1*: $\exists\, x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
from *lifted subset* **have**
linear-order-on A (limit A r) $\wedge$ *linear-order-on A (limit A s)*
using *lifted-def equiv-rel-except-a-def limit-presv-lin-ord*
by *metis*
moreover from *a1 a1-1* **have** *keep-lift*:
$\exists\, x \in A - \{a\}.$ (*let* q = *limit A r in* $a \preceq_q x$) $\wedge$
(*let* u = *limit A s in* $x \preceq_u a$)
using *DiffD1 limit-presv-prefs1*
by *simp*
ultimately show *?thesis*
using *a1 temp*
by (*simp add*: *lifted-def equiv-rel-except-a-def*)
next
assume
$\neg(\exists\, x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a)$
hence *a1-2*:
$\forall\, x \in A - \{a\}.\ \neg(a \preceq_r x \wedge x \preceq_s a)$
by *auto*
moreover have *not-worse*:
$\forall\, x \in A - \{a\}.\ \neg(x \preceq_r a \wedge a \preceq_s x)$
using *lifted subset lifted-mono*
by *fastforce*
moreover have *connex*:
connex A (limit A r) $\wedge$ *connex A (limit A s)*
using *lifted subset lifted-def equiv-rel-except-a-def*
limit-presv-lin-ord lin-ord-imp-connex
by *metis*
moreover have *connex1*:
$\forall\, A\ r.$ *connex A r* =
(*limited A r* $\wedge$ ($\forall\, a.\ (a::'a) \in A \longrightarrow$
($\forall\, aa.\ aa \in A \longrightarrow a \preceq_r aa \vee aa \preceq_r a$)))
by (*simp add*: *Ball-def-raw connex-def*)
hence *limit1*:
limited A (limit A r) $\wedge$
($\forall\, a.\ a \notin A \vee$
($\forall\, aa.$
$aa \notin A \vee (a, aa) \in$ *limit A r* $\vee$
$(aa, a) \in$ *limit A r*))
using *connex connex1*
by *simp*
have *limit2*:
$\forall\, a\ aa\ A\ r.\ (a::'a, aa) \notin$ *limit A r* $\vee\ a \preceq_r aa$
using *limit-presv-prefs2*
by *metis*
have
limited A (limit A s) $\wedge$
($\forall\, a.\ a \notin A \vee$
($\forall\, aa.\ aa \notin A \vee$

$(let\ q = limit\ A\ s\ in\ a \preceq_q aa \vee aa \preceq_q a)))$
using *connex connex-def*
by *metis*
hence *connex2*:
$limited\ A\ (limit\ A\ s) \wedge$
$(\forall a.\ a \notin A \vee$
$(\forall aa.\ aa \notin A \vee$
$((a,\ aa) \in limit\ A\ s \vee (aa,\ a) \in limit\ A\ s)))$
by *simp*
ultimately have
$\forall x \in A - \{a\}.\ (a \preceq_r x \wedge a \preceq_s x) \vee (x \preceq_r a \wedge x \preceq_s a)$
using *DiffD1 limit1 limit-presv-prefs2 a1*
by *metis*
hence *r-eq-s-on-A-0*:
$\forall x \in A - \{a\}.\ ((a,\ x) \in r \wedge (a,\ x) \in s) \vee ((x,\ a) \in r \wedge (x,\ a) \in s)$
by *simp*
have
$\forall x \in A - \{a\}.\ (a,\ x) \in (limit\ A\ r) \longleftrightarrow (a,\ x) \in (limit\ A\ s)$
using *DiffD1 limit2 limit1 connex2 a1 a1-2 not-worse*
by *metis*
hence
$\forall x \in A - \{a\}.$
$(let\ q = limit\ A\ r\ in\ a \preceq_q x) \longleftrightarrow (let\ q = limit\ A\ s\ in\ a \preceq_q x)$
by *simp*
moreover have
$\forall x \in A - \{a\}.\ (x,\ a) \in (limit\ A\ r) \longleftrightarrow (x,\ a) \in (limit\ A\ s)$
using *a1 a1-2 not-worse DiffD1 limit-presv-prefs2 connex2 limit1*
by *metis*
moreover have
$(a,\ a) \in (limit\ A\ r) \wedge (a,\ a) \in (limit\ A\ s)$
using *a1 connex connex-imp-refl refl-onD*
by *metis*
moreover have
$limited\ A\ (limit\ A\ r) \wedge limited\ A\ (limit\ A\ s)$
using *limit-to-limits*
by *metis*
ultimately have
$\forall x\ y.\ (x,\ y) \in limit\ A\ r \longleftrightarrow (x,\ y) \in limit\ A\ s$
using *eql-rs*
by *auto*
thus *?thesis*
by *simp*
qed
next
assume *a2*: $a \notin A$
with *eql-rs* **have**
$\forall x \in A.\ \forall y \in A.\ (x,\ y) \in (limit\ A\ r) \longleftrightarrow (x,\ y) \in (limit\ A\ s)$
by *simp*
thus *?thesis*

using *limit-to-limits limited-dest subrelI subset-antisym*
by *auto*
qed
qed

lemma *negl-diff-imp-eq-limit*:
assumes
change: *equiv-rel-except-a* S r s a **and**
subset: $A \subseteq S$ **and**
notInA: $a \notin A$
shows *limit* A r = *limit* A s
proof –
have $A \subseteq S-\{a\}$
by (*simp add*: *notInA subset subset-Diff-insert*)
hence $\forall x \in A.\ \forall y \in A.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
by (*meson change equiv-rel-except-a-def in-mono*)
thus *?thesis*
by *auto*
qed

theorem *lifted-above-winner*:
assumes
lifted-a: *lifted* A r s a **and**
above-x: *above* r $x = \{x\}$ **and**
fin-A: *finite* A
shows *above* s $x = \{x\} \lor$ *above* s $a = \{a\}$
proof *cases*
assume $x = a$
thus *?thesis*
using *above-subset-geq-one lifted-a above-x*
lifted-above lifted-def equiv-rel-except-a-def
by *metis*
next
assume *asm1*: $x \neq a$
thus *?thesis*
proof *cases*
assume *above* s $x = \{x\}$
thus *?thesis*
by *simp*
next
assume *asm2*: *above* s $x \neq \{x\}$
have $\forall y \in A.\ y \preceq_r x$
proof –
fix *aa* :: $'a$
have *imp-a*: $x \preceq_r aa \longrightarrow aa \notin A \lor aa \preceq_r x$
using *singletonD pref-imp-in-above above-x*
by *metis*
also have *f1*:
$\forall A\ r.$

$(connex\ A\ r \lor$
$(\exists\, a.\ (\exists\, aa.\ \neg\ (aa::'a) \preceq_r a \land \neg\ a \preceq_r aa \land aa \in A) \land a \in A) \lor$
$\neg\ limited\ A\ r) \land$
$((\forall\, a.\ (\forall\, aa.\ aa \preceq_r a \lor a \preceq_r aa \lor aa \notin A) \lor a \notin A) \land limited\ A\ r \lor$
$\neg\ connex\ A\ r)$
using *connex-def*
by *metis*
moreover have *eq-exc-a*:
equiv-rel-except-a A r s a
using *lifted-def lifted-a*
by *metis*
ultimately have $aa \notin A \lor aa \preceq_r x$
using *pref-imp-in-above above-x equiv-rel-except-a-def lin-ord-imp-connex limited-dest insertCI*
by *metis*
thus *?thesis*
using *f1 eq-exc-a above-one above-one2 above-x fin-A equiv-rel-except-a-def insert-not-empty pref-imp-in-above lin-ord-imp-connex mk-disjoint-insert insertE*
by *metis*
qed
moreover have *equiv-rel-except-a A r s a*
using *lifted-a lifted-def*
by *metis*
moreover have $x \in A - \{a\}$
using *above-one above-one2 asm1 assms calculation equiv-rel-except-a-def insert-not-empty member-remove remove-def insert-absorb*
by *metis*
ultimately have $\forall\, y \in A - \{a\}.\ y \preceq_s x$
using *DiffD1 lifted-a equiv-rel-except-a-def*
by *metis*
hence *not-others*: $\forall\, y \in A - \{a\}.\ above\ s\ y \neq \{y\}$
using *asm2 empty-iff insert-iff pref-imp-in-above*
by *metis*
hence $above\ s\ a = \{a\}$
using *Diff-iff all-not-in-conv lifted-a fin-A lifted-def equiv-rel-except-a-def above-one singleton-iff*
by *metis*
thus *?thesis*
by *simp*
qed
qed

theorem *lifted-above-winner2*:
assumes
lifted A r s a **and**
$above\ r\ a = \{a\}$ **and**
finite A

shows *above s a* = {*a*}
using *assms lifted-above-winner*
by *metis*

theorem *lifted-above-winner3*:
assumes
lifted-a: *lifted A r s a* **and**
above-x: *above s x* = {*x*} **and**
fin-A: *finite A* **and**
x-not-a: $x \neq a$
shows *above r x* = {*x*}
proof (*rule ccontr*)
assume *asm*: *above r x* ≠ {*x*}
then obtain *y* **where** *y*: *above r y* = {*y*}
using *lifted-a fin-A insert-Diff insert-not-empty*
lifted-def equiv-rel-except-a-def above-one
by *metis*
hence *above s y* = {*y*} ∨ *above s a* = {*a*}
using *lifted-a fin-A lifted-above-winner*
by *metis*
moreover have ∀ *b*. *above s b* = {*b*} ⟶ *b* = *x*
using *all-not-in-conv lifted-a above-x lifted-def*
fin-A equiv-rel-except-a-def above-one2
by *metis*
ultimately have *y* = *x*
using *x-not-a*
by *presburger*
moreover have $y \neq x$
using *asm y*
by *blast*
ultimately show *False*
by *simp*
qed

end

1.2 Electoral Result

theory *Result*
imports *Main*
begin

An electoral result is the principal result type of the composable modules voting framework, as it is a generalization of the set of winning alternatives from social choice functions. Electoral results are selections of the received (possibly empty) set of alternatives into the three disjoint groups of elected, rejected and deferred alternatives. Any of those sets, e.g., the set of winning (elected) alternatives, may also be left empty, as long as they collectively still hold all the received alternatives.

1.2.1 Definition

type-synonym $'a\ Result = 'a\ set * 'a\ set * 'a\ set$

1.2.2 Auxiliary Functions

fun *disjoint3* :: $'a\ Result \Rightarrow bool$ **where**
  disjoint3 $(e, r, d) =$
    $((e \cap r = \{\}) \wedge$
      $(e \cap d = \{\}) \wedge$
      $(r \cap d = \{\}))$

fun *set-equals-partition* :: $'a\ set \Rightarrow 'a\ Result \Rightarrow bool$ **where**
  set-equals-partition $A\ (e, r, d) = (e \cup r \cup d = A)$

fun *well-formed* :: $'a\ set \Rightarrow 'a\ Result \Rightarrow bool$ **where**
  well-formed A *result* $=$ (*disjoint3 result* $\wedge$ *set-equals-partition* A *result*)

abbreviation *elect-r* :: $'a\ Result \Rightarrow 'a\ set$ **where**
  elect-r $r \equiv$ *fst* r

abbreviation *reject-r* :: $'a\ Result \Rightarrow 'a\ set$ **where**
  reject-r $r \equiv$ *fst* (*snd* r)

abbreviation *defer-r* :: $'a\ Result \Rightarrow 'a\ set$ **where**
  defer-r $r \equiv$ *snd* (*snd* r)

1.2.3 Auxiliary Lemmata

lemma *result-imp-rej*:
  assumes *well-formed* $A\ (e, r, d)$
  shows $A - (e \cup d) = r$
proof (*safe*)
  fix
    $x :: 'a$
  assume
    x-in-A: $x \in A$ **and**
    x-not-rej: $x \notin r$ **and**
    x-not-def: $x \notin d$
  from *assms* **have**
    $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge$
    $(r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
    by *simp*

thus $x \in e$
using *x-in-A x-not-rej x-not-def*
by *auto*
next
fix
$x :: 'a$
assume
x-rej: $x \in r$
from *assms* **have**
$(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge$
$(r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
by *simp*
thus $x \in A$
using *x-rej*
by *auto*
next
fix
$x :: 'a$
assume
x-rej: $x \in r$ **and**
x-elec: $x \in e$
from *assms* **have**
$(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge$
$(r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
by *simp*
thus *False*
using *x-rej x-elec*
by *auto*
next
fix
$x :: 'a$
assume
x-rej: $x \in r$ **and**
x-def: $x \in d$
from *assms* **have**
$(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge$
$(r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
by *simp*
thus *False*
using *x-rej x-def*
by *auto*
qed

lemma *result-count*:
assumes
well-formed A (e, r, d) **and**
finite A
shows *card* A = *card* e + *card* r + *card* d
proof $-$

from *assms*(*1*) **have** *disj*:
$(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\})$
by *simp*
from *assms*(*1*) **have** *set-partit*:
$e \cup r \cup d = A$
by *simp*
show *?thesis*
using *assms disj set-partit Int-Un-distrib2 finite-Un card-Un-disjoint sup-bot.right-neutral*
by *metis*
qed

end

1.3 Preference Profile

theory *Profile*
imports *Preference-Relation*
begin

Preference profiles denote the decisions made by the individual voters on the eligible alternatives. They are represented in the form of one preference relation (e.g., selected on a ballot) per voter, collectively captured in a list of such preference relations. Unlike a the common preference profiles in the social-choice sense, the profiles described here considers only the (sub-)set of alternatives that are received.

1.3.1 Definition

type-synonym $'a$ *Profile* = ($'a$ *Preference-Relation*) *list*

definition *profile* :: $'a$ *set* ⇒ $'a$ *Profile* ⇒ *bool* **where**
profile A p ≡ ∀ *i*::*nat*. *i* < *size p* ⟶ *linear-order-on A* (*p*!*i*)

lemma *profile-set* : *profile A p* ≡ (∀ *b* ∈ (*set p*). *linear-order-on A b*)
by (*simp add*: *all-set-conv-all-nth profile-def*)

abbreviation *finite-profile* :: $'a$ *set* ⇒ $'a$ *Profile* ⇒ *bool* **where**
finite-profile A p ≡ *finite A* ∧ *profile A p*

1.3.2 Preference Counts and Comparisons

```
fun win-count :: 'a Profile ⇒ 'a ⇒ nat where
  win-count p a =
    card {i::nat. i < size p ∧ above (p!i) a = {a}}

fun win-count-code :: 'a Profile ⇒ 'a ⇒ nat where
  win-count-code Nil a = 0 |
  win-count-code (p#ps) a =
      (if (above p a = {a}) then 1 else 0) + win-count-code ps a

fun prefer-count :: 'a Profile ⇒ 'a ⇒ 'a ⇒ nat where
  prefer-count p x y =
      card {i::nat. i < size p ∧ (let r = (p!i) in (y ≼r x))}

fun prefer-count-code :: 'a Profile ⇒ 'a ⇒ 'a ⇒ nat where
  prefer-count-code Nil x y = 0 |
  prefer-count-code (p#ps) x y =
      (if y ≼p x then 1 else 0) + prefer-count-code ps x y

lemma set-compr: { f x | x . x ∈ S } = f ` S
  by auto

lemma pref-count-set-compr: {prefer-count p x y | y . y ∈ A−{x}} =
        (prefer-count p x) ` (A−{x})
  by auto

lemma pref-count:
  assumes prof: profile A p
  assumes x-in-A: x ∈ A
  assumes y-in-A: y ∈ A
  assumes neq: x ≠ y
  shows prefer-count p x y = (size p) − (prefer-count p y x)
proof −
  have 00: card {i::nat. i < size p} = size p
    by simp
  have 10:
    {i::nat. i < size p ∧ (let r = (p!i) in (y ≼r x))} =
        {i::nat. i < size p} −
          {i::nat. i < size p ∧ ¬ (let r = (p!i) in (y ≼r x))}
    by auto
  have 0: ∀ i::nat . i < size p ⟶ linear-order-on A (p!i)
    using prof profile-def
    by metis
  hence ∀ i::nat . i < size p ⟶ connex A (p!i)
    by (simp add: lin-ord-imp-connex)
  hence 1: ∀ i::nat . i < size p ⟶
            ¬ (let r = (p!i) in (y ≼r x)) ⟶ (let r = (p!i) in (x ≼r y))
    using connex-def x-in-A y-in-A
    by metis
```

from *0* **have**
$\forall\ i{::}nat\ .\ i < size\ p \longrightarrow antisym\ \ (p!i)$
using *lin-imp-antisym*
by *metis*
hence $\forall\ i{::}nat\ .\ i < size\ p \longrightarrow ((y,\ x) \in (p!i) \longrightarrow (x,\ y) \notin (p!i))$
using *antisymD neq*
by *metis*
hence $\forall\ i{::}nat\ .\ i < size\ p \longrightarrow$
$((let\ r = (p!i)\ in\ (y \preceq_r x)) \longrightarrow \neg\ (let\ r = (p!i)\ in\ (x \preceq_r y)))$
by *simp*
with *1* **have**
$\forall\ i{::}nat\ .\ i < size\ p \longrightarrow$
$\neg\ (let\ r = (p!i)\ in\ (y \preceq_r x)) = (let\ r = (p!i)\ in\ (x \preceq_r y))$
by *metis*
hence *2*:
$\{i{::}nat.\ i < size\ p \wedge \neg\ (let\ r = (p!i)\ in\ (y \preceq_r x))\} =$
$\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\}$
by *metis*
hence *20*:
$\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r x))\} =$
$\{i{::}nat.\ i < size\ p\} - \{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\}$
using *10 2*
by *simp*
have $\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\} \subseteq \{i{::}nat.\ i < size\ p\}$
by (*simp add*: *Collect-mono*)
hence *30*:
$card\ (\{i{::}nat.\ i < size\ p\} -$
$\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\}) =$
$(card\ \{i{::}nat.\ i < size\ p\}) -$
$card(\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\})$
by (*simp add*: *card-Diff-subset*)
have *prefer-count p x y* $=$
$card\ \{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r x))\}$
by *simp*
also have
$... = card(\{i{::}nat.\ i < size\ p\} -$
$\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\})$
using *20*
by *simp*
also have
$... = (card\ \{i{::}nat.\ i < size\ p\}) -$
$card(\{i{::}nat.\ i < size\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\})$
using *30*
by *metis*
also have
$... = size\ p - (prefer\text{-}count\ p\ y\ x)$
by *simp*
finally show *?thesis*
by (*simp add*: *20 30*)

qed

lemma *pref-count-sym*:
assumes *p1*: *prefer-count p a x* $\geq$ *prefer-count p x b*
assumes *prof*: *profile A p*
assumes *a-in-A*: $a \in A$
assumes *b-in-A*: $b \in A$
assumes *x-in-A*: $x \in A$
assumes *neq1*: $a \neq x$
assumes *neq2*: $x \neq b$
shows *prefer-count p b x* $\geq$ *prefer-count p x a*
proof −
from *prof a-in-A x-in-A neq1* **have** *0*:
prefer-count p a x = (*size p*) − (*prefer-count p x a*)
using *pref-count*
by *metis*
moreover from *prof x-in-A b-in-A neq2* **have** *1*:
prefer-count p x b = (*size p*) − (*prefer-count p b x*)
using *pref-count*
by (*metis* (*mono-tags*, *lifting*))
hence *2*: (*size p*) − (*prefer-count p x a*) $\geq$
(*size p*) − (*prefer-count p b x*)
using *calculation p1*
by *auto*
hence *3*: (*prefer-count p x a*) − (*size p*) $\leq$
(*prefer-count p b x*) − (*size p*)
using *a-in-A diff-is-0-eq diff-le-self neq1*
pref-count prof x-in-A
by (*metis* (*no-types*))
hence (*prefer-count p x a*) $\leq$ (*prefer-count p b x*)
using *1 3 calculation p1*
by *linarith*
thus *?thesis*
by *linarith*
qed

lemma *empty-prof-imp-zero-pref-count*:
assumes $p = []$
shows $\forall$ *x y*. *prefer-count p x y* = *0*
using *assms*
by *simp*

lemma *pref-count-code-incr*:
assumes
prefer-count-code ps x y = *n* **and**
$y \preceq_p x$
shows *prefer-count-code* (*p#ps*) *x y* = *n+1*
using *assms*
by *simp*

lemma *pref-count-code-not-smaller-imp-constant*:
assumes
prefer-count-code ps x y = *n* **and**
$\neg(y \preceq_p x)$
shows *prefer-count-code* (*p*#*ps*) *x y* = *n*
using *assms*
by *simp*

fun *wins* :: $'a \Rightarrow 'a$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
wins x p y =
(*prefer-count p x y* > *prefer-count p y x*)

fun *wins-code* :: $'a \Rightarrow 'a$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
wins-code x p y =
(*prefer-count-code p x y* > *prefer-count-code p y x*)

lemma *wins-antisym*:
assumes *wins a p b*
shows ¬ *wins b p a*
using *assms*
by *simp*

lemma *wins-irreflex*: ¬ *wins w p w*
using *wins-antisym*
by *metis*

1.3.3 Condorcet Winner

fun *condorcet-winner* :: $'a$ *set* $\Rightarrow 'a$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
condorcet-winner A p w =
(*finite-profile A p* ∧ $w \in A \wedge (\forall x \in A - \{w\}$. *wins w p x*))

fun *condorcet-winner-code* :: $'a$ *set* $\Rightarrow 'a$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
condorcet-winner-code A p w =
(*finite-profile A p* ∧ $w \in A \wedge$
$(\forall x \in A - \{w\}$. *wins-code w p x*))

lemma *cond-winner-unique*:
assumes *winner-c*: *condorcet-winner A p c* **and**
winner-w: *condorcet-winner A p w*
shows *w* = *c*
proof (*rule ccontr*)
assume
assumption: $w \neq c$
from *winner-w*
have *wins w p c*
using *assumption insert-Diff insert-iff winner-c*

```
    by simp
  hence ¬ wins c p w
    by (simp add: wins-antisym)
  moreover from winner-c
  have
    c-wins-against-w: wins c p w
    using Diff-iff assumption
          singletonD winner-w
    by simp
  ultimately show False
    by simp
qed

lemma cond-winner-unique2:
  assumes winner: condorcet-winner A p w and
          not-w:  x ≠ w and
          in-A:   x ∈ A
        shows ¬ condorcet-winner A p x
  using not-w cond-winner-unique winner
  by metis

lemma cond-winner-unique3:
  assumes condorcet-winner A p w
  shows {a ∈ A. condorcet-winner A p a} = {w}
proof (safe, simp-all, safe)
  fix
    x :: 'a
  assume
    fin-A: finite A and
    prof-A: profile A p and
    x-in-A: x ∈ A and
    x-wins:
      ∀ xa ∈ A − {x}.
        card {i. i < size p ∧ (x, xa) ∈ p!i} <
          card {i. i < size p ∧ (xa, x) ∈ p!i}
  from assms have assm:
    finite-profile A p ∧  w ∈ A ∧
      (∀ x ∈ A − {w}.
        card {i::nat. i < size p ∧ (w, x) ∈ p!i} <
          card {i::nat. i < size p ∧ (x, w) ∈ p!i})
    by simp
  hence
    (∀ x ∈ A − {w}.
      card {i::nat. i < size p ∧ (w, x) ∈ p!i} <
        card {i::nat. i < size p ∧ (x, w) ∈ p!i})
    by simp
  hence w-beats-x:
    x ≠ w ⟹
      card {i::nat. i < size p ∧ (w, x) ∈ p!i} <
```

```
      card {i::nat. i < size p ∧ (x, w) ∈ p!i}
    using x-in-A
    by simp
  also from assm have
    finite-profile A p
    by simp
  moreover from assm have
    w ∈ A
    by simp
  hence x-beats-w:
    x ≠ w ⟹
      card {i. i < size p ∧ (x, w) ∈ p!i} <
        card {i. i < size p ∧ (w, x) ∈ p!i}
    using x-wins
    by simp
  from w-beats-x x-beats-w show
    x = w
    by linarith
next
  fix
    x :: 'a
  from assms show w ∈ A
    by simp
next
  fix
    x :: 'a
  from assms show finite A
    by simp
next
  fix
    x :: 'a
  from assms show profile A p
    by simp
next
  fix
    x :: 'a
  from assms show w ∈ A
    by simp
next
  fix
    x :: 'a and
    xa :: 'a
  assume
    xa-in-A: xa ∈ A and
    w-wins:
      ¬ card {i. i < length p ∧ (w, xa) ∈ p!i} <
        card {i. i < length p ∧ (xa, w) ∈ p!i}
  from assms have
    finite-profile A p ∧  w ∈ A ∧
```

($\forall x \in A - \{w\}$.
  card $\{i{::}nat.\ i < size\ p \land (w, x) \in p!i\} <$
    card $\{i{::}nat.\ i < size\ p \land (x, w) \in p!i\})$
by *simp*
thus $xa = w$
  using *xa-in-A w-wins insert-Diff insert-iff*
  by (*metis* (*no-types*, *lifting*))
qed

1.3.4 Limited Profile

fun *limit-profile* :: *'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$ *'a Profile* **where**
  limit-profile A p = *map* (*limit A*) *p*

lemma *limit-prof-trans*:
  assumes
    $B \subseteq A$ **and**
    $C \subseteq B$ **and**
    finite-profile A p
  shows *limit-profile C p* = *limit-profile C* (*limit-profile B p*)
  using *assms*
  by *auto*

lemma *limit-profile-sound*:
  assumes
    profile: *finite-profile S p* **and**
    subset: $A \subseteq S$
  shows *finite-profile A* (*limit-profile A p*)
proof (*simp*)
  from *profile*
  show *finite-profile A* (*map* (*limit A*) *p*)
    using *length-map limit-presv-lin-ord nth-map*
        profile-def subset infinite-super
    by *metis*
qed

lemma *limit-prof-presv-size*:
  assumes *f-prof*: *finite-profile S p* **and**
        subset: $A \subseteq S$
  shows *size p* = *size* (*limit-profile A p*)
  by *simp*

1.3.5 Lifting Property

definition *equiv-prof-except-a* :: *'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$ *'a Profile* $\Rightarrow$
                'a $\Rightarrow$ *bool* **where**
  equiv-prof-except-a A p q a $\equiv$
    finite-profile A p $\land$ *finite-profile A q* $\land$ $a \in A$ $\land$ *size p* = *size q* $\land$
    ($\forall i{::}nat$.
      $i <$ *size p* $\longrightarrow$

equiv-rel-except-a A $(p!i)$ $(q!i)$ *a*)

definition *lifted* :: *'a set* ⇒ *'a Profile* ⇒ *'a Profile* ⇒ *'a* ⇒ *bool* **where**
lifted A p q a ≡
finite-profile A p ∧ *finite-profile A q* ∧ *a* ∈ *A* ∧ *size p* = *size q* ∧
(∀ *i*::*nat*.
(*i* < *size p* ∧ ¬*Preference-Relation.lifted A* $(p!i)$ $(q!i)$ *a*) ⟶
$(p!i) = (q!i)$) ∧
(∃ *i*::*nat*. *i* < *size p* ∧ *Preference-Relation.lifted A* $(p!i)$ $(q!i)$ *a*)

lemma *lifted-imp-equiv-prof-except-a*:
assumes *lifted*: *lifted A p q a*
shows *equiv-prof-except-a A p q a*
proof −
have
∀ *i*::*nat*. *i* < *size p* ⟶
equiv-rel-except-a A $(p!i)$ $(q!i)$ *a*
proof
fix *i* :: *nat*
show
i < *size p* ⟶
equiv-rel-except-a A $(p!i)$ $(q!i)$ *a*
proof
assume *i-ok*: *i* < *size p*
show *equiv-rel-except-a A* $(p!i)$ $(q!i)$ *a*
using *lifted-def i-ok lifted profile-def trivial-equiv-rel*
lifted-imp-equiv-rel-except-a
by *metis*
qed
qed
thus *?thesis*
using *lifted-def lifted equiv-prof-except-a-def*
by *metis*
qed

lemma *negl-diff-imp-eq-limit-prof*:
assumes
change: *equiv-prof-except-a S p q a* **and**
subset: *A* ⊆ *S* **and**
notInA: *a* ∉ *A*
shows *limit-profile A p* = *limit-profile A q*
proof −
have
∀ *i*::*nat*. *i* < *size p* ⟶
equiv-rel-except-a S $(p!i)$ $(q!i)$ *a*
using *change equiv-prof-except-a-def*
by *metis*
hence ∀ *i*::*nat*. *i* < *size p* ⟶ *limit A* $(p!i)$ = *limit A* $(q!i)$

using *notInA negl-diff-imp-eq-limit subset*
by *metis*
hence *map* (*limit* A) p = *map* (*limit* A) q
using *change equiv-prof-except-a-def*
length-map nth-equalityI nth-map
by (*metis* (*mono-tags*, *lifting*))
thus *?thesis*
by *simp*
qed

lemma *limit-prof-eq-or-lifted*:
assumes
lifted: *lifted* S p q a **and**
subset: $A \subseteq S$
shows
limit-profile A p = *limit-profile* A q $\vee$
lifted A (*limit-profile* A p) (*limit-profile* A q) a
proof *cases*
assume *inA*: $a \in A$
have
$\forall$ *i::nat*. $i <$ *size* $p \longrightarrow$
(*Preference-Relation.lifted* S ($p!i$) ($q!i$) $a \vee$ ($p!i$) = ($q!i$))
using *lifted-def lifted*
by *metis*
hence *one*:
$\forall$ *i::nat*. $i <$ *size* $p \longrightarrow$
(*Preference-Relation.lifted* A (*limit* A ($p!i$)) (*limit* A ($q!i$)) $a \vee$
(*limit* A ($p!i$)) = (*limit* A ($q!i$)))
using *limit-lifted-imp-eq-or-lifted subset*
by *metis*
thus *?thesis*
proof *cases*
assume $\forall$ *i::nat*. $i <$ *size* $p \longrightarrow$ (*limit* A ($p!i$)) = (*limit* A ($q!i$))
thus *?thesis*
using *lifted-def length-map lifted*
limit-profile.simps nth-equalityI nth-map
by (*metis* (*mono-tags*, *lifting*))
next
assume *assm*:
$\neg(\forall$ *i::nat*. $i <$ *size* $p \longrightarrow$ (*limit* A ($p!i$)) = (*limit* A ($q!i$)))
let *?p* = *limit-profile* A p
let *?q* = *limit-profile* A q
have *profile* A *?p* $\wedge$ *profile* A *?q*
using *lifted-def lifted limit-profile-sound subset*
by *metis*
moreover have *size ?p* = *size ?q*
using *lifted-def lifted*
by *fastforce*
moreover have

$\exists\, i{::}nat.\ i < size\ ?p \wedge Preference\text{-}Relation.lifted\ A\ (?p!i)\ (?q!i)\ a$
    using *assm lifted-def length-map lifted*
        limit-profile.simps nth-map one
    by (*metis* (*no-types*, *lifting*))
  moreover have
    $\forall\, i{::}nat.$
      $(i < size\ ?p \wedge \neg Preference\text{-}Relation.lifted\ A\ (?p!i)\ (?q!i)\ a) \longrightarrow$
        $(?p!i) = (?q!i)$
    using *lifted-def length-map lifted*
        limit-profile.simps nth-map one
    by *metis*
  ultimately have *lifted A ?p ?q a*
    using *lifted-def inA lifted rev-finite-subset subset*
    by (*metis* (*no-types*, *lifting*))
  thus *?thesis*
    by *simp*
 qed
next
 assume $a \notin A$
 thus *?thesis*
  using *lifted negl-diff-imp-eq-limit-prof subset*
      lifted-imp-equiv-prof-except-a
  by *metis*
qed

end

Chapter 2

Component Types

2.1 Electoral Module

theory *Electoral-Module*
  imports *Social-Choice-Types/Preference-Relation*
    Social-Choice-Types/Profile
    Social-Choice-Types/Result
begin

Electoral modules are the principal component type of the composable modules voting framework, as they are a generalization of voting rules in the sense of social choice functions. These are only the types used for electoral modules. Further restrictions are encompassed by the electoral-module predicate.

An electoral module does not need to make final decisions for all alternatives, but can instead defer the decision for some or all of them to other modules. Hence, electoral modules partition the received (possibly empty) set of alternatives into elected, rejected and deferred alternatives. In particular, any of those sets, e.g., the set of winning (elected) alternatives, may also be left empty, as long as they collectively still hold all the received alternatives. Just like a voting rule, an electoral module also receives a profile which holds the voters preferences, which, unlike a voting rule, consider only the (sub-)set of alternatives that the module receives.

2.1.1 Definition

type-synonym $'a$ *Electoral-Module* $=$ $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Result*

2.1.2 Auxiliary Definitions

definition *electoral-module* :: $'a$ *Electoral-Module* $\Rightarrow$ *bool* **where**
  electoral-module $m \equiv \forall\, A\ p.$ *finite-profile* $A\ p \longrightarrow$ *well-formed* $A\ (m\ A\ p)$

lemma *electoral-modI*:
  $((\bigwedge A\ p.\ \llbracket\ \textit{finite-profile}\ A\ p\ \rrbracket \Longrightarrow \textit{well-formed}\ A\ (m\ A\ p)) \Longrightarrow$
    $\textit{electoral-module}\ m)$
  unfolding *electoral-module-def*
  by *auto*

abbreviation *elect* ::
  $'a\ \textit{Electoral-Module} \Rightarrow\ 'a\ \textit{set} \Rightarrow\ 'a\ \textit{Profile} \Rightarrow\ 'a\ \textit{set}$ **where**
  $\textit{elect}\ m\ A\ p \equiv \textit{elect-r}\ (m\ A\ p)$

abbreviation *reject* ::
  $'a\ \textit{Electoral-Module} \Rightarrow\ 'a\ \textit{set} \Rightarrow\ 'a\ \textit{Profile} \Rightarrow\ 'a\ \textit{set}$ **where**
  $\textit{reject}\ m\ A\ p \equiv \textit{reject-r}\ (m\ A\ p)$

abbreviation *defer* ::
  $'a\ \textit{Electoral-Module} \Rightarrow\ 'a\ \textit{set} \Rightarrow\ 'a\ \textit{Profile} \Rightarrow\ 'a\ \textit{set}$ **where**
  $\textit{defer}\ m\ A\ p \equiv \textit{defer-r}\ (m\ A\ p)$

definition *defers* :: $\textit{nat} \Rightarrow\ 'a\ \textit{Electoral-Module} \Rightarrow \textit{bool}$ **where**
  $\textit{defers}\ n\ m \equiv$
    $\textit{electoral-module}\ m\ \wedge$
      $(\forall\ A\ p.\ (\textit{card}\ A \geq n \wedge \textit{finite-profile}\ A\ p) \longrightarrow$
        $\textit{card}\ (\textit{defer}\ m\ A\ p) = n)$

definition *rejects* :: $\textit{nat} \Rightarrow\ 'a\ \textit{Electoral-Module} \Rightarrow \textit{bool}$ **where**
  $\textit{rejects}\ n\ m \equiv$
    $\textit{electoral-module}\ m\ \wedge$
      $(\forall\ A\ p.\ (\textit{card}\ A \geq n \wedge \textit{finite-profile}\ A\ p) \longrightarrow \textit{card}\ (\textit{reject}\ m\ A\ p) = n)$

definition *eliminates* :: $\textit{nat} \Rightarrow\ 'a\ \textit{Electoral-Module} \Rightarrow \textit{bool}$ **where**
  $\textit{eliminates}\ n\ m \equiv$
    $\textit{electoral-module}\ m\ \wedge$
      $(\forall\ A\ p.\ (\textit{card}\ A > n \wedge \textit{finite-profile}\ A\ p) \longrightarrow \textit{card}\ (\textit{reject}\ m\ A\ p) = n)$

definition *elects* :: $\textit{nat} \Rightarrow\ 'a\ \textit{Electoral-Module} \Rightarrow \textit{bool}$ **where**
  $\textit{elects}\ n\ m \equiv$
    $\textit{electoral-module}\ m\ \wedge$
      $(\forall\ A\ p.\ (\textit{card}\ A \geq n \wedge \textit{finite-profile}\ A\ p) \longrightarrow \textit{card}\ (\textit{elect}\ m\ A\ p) = n)$

definition *indep-of-alt* :: $'a\ \textit{Electoral-Module} \Rightarrow\ 'a\ \textit{set} \Rightarrow\ 'a \Rightarrow \textit{bool}$ **where**
  $\textit{indep-of-alt}\ m\ A\ a \equiv$

electoral-module $m \wedge (\forall\, p\ q.\ \textit{equiv-prof-except-a}\ A\ p\ q\ a \longrightarrow m\ A\ p = m\ A\ q)$

2.1.3 Equivalence Definitions

definition *prof-contains-result* :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$
$'a$ *Profile* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
prof-contains-result $m\ A\ p\ q\ a \equiv$
electoral-module $m \wedge$ *finite-profile* $A\ p \wedge$ *finite-profile* $A\ q \wedge a \in A \wedge$
$(a \in \textit{elect}\ m\ A\ p \longrightarrow a \in \textit{elect}\ m\ A\ q) \wedge$
$(a \in \textit{reject}\ m\ A\ p \longrightarrow a \in \textit{reject}\ m\ A\ q) \wedge$
$(a \in \textit{defer}\ m\ A\ p \longrightarrow a \in \textit{defer}\ m\ A\ q)$

definition *prof-leq-result* :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$
$'a$ *Profile* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
prof-leq-result $m\ A\ p\ q\ a \equiv$
electoral-module $m \wedge$ *finite-profile* $A\ p \wedge$ *finite-profile* $A\ q \wedge a \in A \wedge$
$(a \in \textit{reject}\ m\ A\ p \longrightarrow a \in \textit{reject}\ m\ A\ q) \wedge$
$(a \in \textit{defer}\ m\ A\ p \longrightarrow a \notin \textit{elect}\ m\ A\ q)$

definition *prof-geq-result* :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$
$'a$ *Profile* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
prof-geq-result $m\ A\ p\ q\ a \equiv$
electoral-module $m \wedge$ *finite-profile* $A\ p \wedge$ *finite-profile* $A\ q \wedge a \in A \wedge$
$(a \in \textit{elect}\ m\ A\ p \longrightarrow a \in \textit{elect}\ m\ A\ q) \wedge$
$(a \in \textit{defer}\ m\ A\ p \longrightarrow a \notin \textit{reject}\ m\ A\ q)$

definition *mod-contains-result* :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *Electoral-Module* $\Rightarrow$
$'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
mod-contains-result $m\ n\ A\ p\ a \equiv$
electoral-module $m \wedge$ *electoral-module* $n \wedge$ *finite-profile* $A\ p \wedge a \in A \wedge$
$(a \in \textit{elect}\ m\ A\ p \longrightarrow a \in \textit{elect}\ n\ A\ p) \wedge$
$(a \in \textit{reject}\ m\ A\ p \longrightarrow a \in \textit{reject}\ n\ A\ p) \wedge$
$(a \in \textit{defer}\ m\ A\ p \longrightarrow a \in \textit{defer}\ n\ A\ p)$

2.1.4 Auxiliary Lemmata

lemma *combine-ele-rej-def*:
assumes
ele: $\textit{elect}\ m\ A\ p = e$ **and**
rej: $\textit{reject}\ m\ A\ p = r$ **and**
def: $\textit{defer}\ m\ A\ p = d$
shows $m\ A\ p = (e,\ r,\ d)$
using *def ele rej*
by *auto*

lemma *par-comp-result-sound*:
assumes
mod-m: *electoral-module* m **and**
f-prof: *finite-profile* $A\ p$
shows *well-formed* $A\ (m\ A\ p)$

  using *electoral-module-def mod-m f-prof*
  by *auto*

lemma *result-presv-alts*:
  assumes
    e-mod: *electoral-module* m **and**
    f-prof: *finite-profile* A p
  shows $(elect\ m\ A\ p) \cup (reject\ m\ A\ p) \cup (defer\ m\ A\ p) = A$
proof (*safe*)
  fix
    $x :: 'a$
  assume
    asm: $x \in elect\ m\ A\ p$
  have *partit*:
    $\forall A\ p.$
      $\neg$ *set-equals-partition* $(A::'a\ set)\ p \vee$
        $(\exists B\ C\ D\ E.\ A = B \wedge p = (C, D, E) \wedge C \cup D \cup E = B)$
    by *simp*
  from *e-mod f-prof* **have** *set-partit*:
    set-equals-partition A $(m\ A\ p)$
    using *electoral-module-def*
    by *auto*
  thus $x \in A$
    using *UnI1 asm fstI set-partit partit*
    by (*metis* (*no-types*))
next
  fix
    $x :: 'a$
  assume
    asm: $x \in reject\ m\ A\ p$
  have *partit*:
    $\forall A\ p.$
      $\neg$ *set-equals-partition* $(A::'a\ set)\ p \vee$
        $(\exists B\ C\ D\ E.\ A = B \wedge p = (C, D, E) \wedge C \cup D \cup E = B)$
    by *simp*
  from *e-mod f-prof* **have** *set-partit*:
    set-equals-partition A $(m\ A\ p)$
    using *electoral-module-def*
    by *auto*
  thus $x \in A$
    using *UnI1 asm fstI set-partit partit*
        sndI subsetD sup-ge2
    by *metis*
next
  fix
    $x :: 'a$
  assume
    asm: $x \in defer\ m\ A\ p$
  have *partit*:

$\forall A\ p.$
  $\neg$ *set-equals-partition* $(A::'a\ set)\ p \lor$
    $(\exists B\ C\ D\ E.\ A = B \land p = (C, D, E) \land C \cup D \cup E = B)$
  by *simp*
from *e-mod f-prof* **have** *set-partit*:
  set-equals-partition $A\ (m\ A\ p)$
  using *electoral-module-def*
  by *auto*
thus $x \in A$
  using *asm set-partit partit sndI subsetD sup-ge2*
  by *metis*

next
fix
  $x :: 'a$
assume
  asm1: $x \in A$ **and**
  asm2: $x \notin$ *defer* $m\ A\ p$ **and**
  asm3: $x \notin$ *reject* $m\ A\ p$
have *partit*:
  $\forall A\ p.$
    $\neg$ *set-equals-partition* $(A::'a\ set)\ p \lor$
      $(\exists B\ C\ D\ E.\ A = B \land p = (C, D, E) \land C \cup D \cup E = B)$
  by *simp*
from *e-mod f-prof* **have** *set-partit*:
  set-equals-partition $A\ (m\ A\ p)$
  using *electoral-module-def*
  by *auto*
show $x \in$ *elect* $m\ A\ p$
  using *asm1 asm2 asm3 fst-conv partit*
    set-partit snd-conv Un-iff
  by *metis*

qed

lemma *result-disj*:
assumes
  module: *electoral-module* m **and**
  profile: *finite-profile* $A\ p$
shows
  $(\textit{elect}\ m\ A\ p) \cap (\textit{reject}\ m\ A\ p) = \{\} \land$
    $(\textit{elect}\ m\ A\ p) \cap (\textit{defer}\ m\ A\ p) = \{\} \land$
    $(\textit{reject}\ m\ A\ p) \cap (\textit{defer}\ m\ A\ p) = \{\}$

proof (*safe*, *simp-all*)
fix
  $x :: 'a$
assume
  asm1: $x \in$ *elect* $m\ A\ p$ **and**
  asm2: $x \in$ *reject* $m\ A\ p$
have *partit*:
  $\forall A\ p.$

$\neg$ *set-equals-partition* $(A::'a\ set)\ p \lor$
$(\exists B\ C\ D\ E.\ A = B \land p = (C, D, E) \land C \cup D \cup E = B)$
by *simp*
from *module profile* **have** *set-partit*:
set-equals-partition $A\ (m\ A\ p)$
using *electoral-module-def*
by *auto*
from *profile* **have** *prof-p*:
finite $A \land$ *profile* $A\ p$
by *simp*
from *module prof-p* **have** *wf-A-m*:
well-formed $A\ (m\ A\ p)$
using *electoral-module-def*
by *metis*
show *False*
using *prod.exhaust-sel DiffE UnCI asm1 asm2*
module profile result-imp-rej wf-A-m
prof-p set-partit partit
by (*metis* (*no-types*))
next
fix
$x :: 'a$
assume
asm1: $x \in$ *elect* $m\ A\ p$ **and**
asm2: $x \in$ *defer* $m\ A\ p$
have *partit*:
$\forall A\ p.$
$\neg$ *set-equals-partition* $(A::'a\ set)\ p \lor$
$(\exists B\ C\ D\ E.\ A = B \land p = (C, D, E) \land C \cup D \cup E = B)$
by *simp*
have *disj*:
$\forall p.\ \neg$ *disjoint3* $p \lor$
$(\exists B\ C\ D.\ p = (B::'a\ set, C, D) \land$
$B \cap C = \{\} \land B \cap D = \{\} \land C \cap D = \{\})$
by *simp*
from *profile* **have** *prof-p*:
finite $A \land$ *profile* $A\ p$
by *simp*
from *module prof-p* **have** *wf-A-m*:
well-formed $A\ (m\ A\ p)$
using *electoral-module-def*
by *metis*
hence *wf-A-m-0*:
disjoint3 $(m\ A\ p) \land$ *set-equals-partition* $A\ (m\ A\ p)$
by *simp*
hence *disj3*:
disjoint3 $(m\ A\ p)$
by *simp*
have *set-partit*:

set-equals-partition A (*m A p*)
  using *wf-A-m-0*
  by *simp*
 from *disj3* **obtain**
  AA :: *'a Result* ⇒ *'a set* **and**
  AAa :: *'a Result* ⇒ *'a set* **and**
  AAb :: *'a Result* ⇒ *'a set*
  where
  m A p =
   (*AA* (*m A p*), *AAa* (*m A p*), *AAb* (*m A p*)) ∧
    AA (*m A p*) ∩ *AAa* (*m A p*) = {} ∧
    AA (*m A p*) ∩ *AAb* (*m A p*) = {} ∧
    AAa (*m A p*) ∩ *AAb* (*m A p*) = {}
  using *asm1 asm2 disj*
  by *metis*
 hence ((*elect m A p*) ∩ (*reject m A p*) = {}) ∧
    ((*elect m A p*) ∩ (*defer m A p*) = {}) ∧
    ((*reject m A p*) ∩ (*defer m A p*) = {})
  using *disj3 eq-snd-iff fstI*
  by *metis*
 thus *False*
  using *asm1 asm2 module profile wf-A-m prof-p*
    set-partit partit disjoint-iff-not-equal
  by (*metis* (*no-types*))
next
 fix
  x :: *'a*
 assume
  asm1: *x* ∈ *reject m A p* **and**
  asm2: *x* ∈ *defer m A p*
 have *partit*:
  ∀ *A p*.
   ¬ *set-equals-partition* (*A*::*'a set*) *p* ∨
    (∃ *B C D E*. *A* = *B* ∧ *p* = (*C*, *D*, *E*) ∧ *C* ∪ *D* ∪ *E* = *B*)
  by *simp*
 from *module profile* **have** *set-partit*:
  set-equals-partition A (*m A p*)
  using *electoral-module-def*
  by *auto*
 from *profile* **have** *prof-p*:
  finite A ∧ *profile A p*
  by *simp*
 from *module prof-p* **have** *wf-A-m*:
  well-formed A (*m A p*)
  using *electoral-module-def*
  by *metis*
 show *False*
  using *prod.exhaust-sel DiffE UnCI asm1 asm2*
    module profile result-imp-rej wf-A-m

prof-p set-partit partit
by (*metis* (*no-types*))
qed

lemma *elect-in-alts*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *elect m A p* $\subseteq$ *A*
using *le-supI1 e-mod f-prof result-presv-alts sup-ge1*
by *metis*

lemma *reject-in-alts*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *reject m A p* $\subseteq$ *A*
using *le-supI1 e-mod f-prof result-presv-alts sup-ge2*
by *fastforce*

lemma *defer-in-alts*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *defer m A p* $\subseteq$ *A*
using *e-mod f-prof result-presv-alts*
by *auto*

lemma *def-presv-fin-prof*:
assumes *module*: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows
let new-A = *defer m A p in*
finite-profile new-A (*limit-profile new-A p*)
using *defer-in-alts infinite-super*
limit-profile-sound module f-prof
by *metis*

lemma *upper-card-bounds-for-result*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows
card (*elect m A p*) $\leq$ *card A* $\wedge$
card (*reject m A p*) $\leq$ *card A* $\wedge$
card (*defer m A p*) $\leq$ *card A*
by (*simp add*: *card-mono defer-in-alts elect-in-alts*
e-mod f-prof reject-in-alts)

lemma *reject-not-elec-or-def*:
  assumes
    e-mod: *electoral-module m* **and**
    f-prof: *finite-profile A p*
  shows *reject m A p* = *A* − (*elect m A p*) − (*defer m A p*)
proof −
  from *e-mod f-prof* **have** *0*: *well-formed A* (*m A p*)
    by (*simp add*: *electoral-module-def*)
  with *e-mod f-prof*
    have (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
      using *result-presv-alts*
      by *simp*
    moreover from *0* **have**
      (*elect m A p*) ∩ (*reject m A p*) = {} ∧
        (*reject m A p*) ∩ (*defer m A p*) = {}
    using *e-mod f-prof result-disj*
    by *blast*
  ultimately show *?thesis*
    by *blast*
qed

lemma *elec-and-def-not-rej*:
  assumes
    e-mod: *electoral-module m* **and**
    f-prof: *finite-profile A p*
  shows *elect m A p* ∪ *defer m A p* = *A* − (*reject m A p*)
proof −
  from *e-mod f-prof* **have** *0*: *well-formed A* (*m A p*)
    by (*simp add*: *electoral-module-def*)
  hence
    disjoint3 (*m A p*) ∧ *set-equals-partition A* (*m A p*)
    by *simp*
  with *e-mod f-prof*
  have (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
    using *e-mod f-prof result-presv-alts*
    by *blast*
  moreover from *0* **have**
    (*elect m A p*) ∩ (*reject m A p*) = {} ∧
      (*reject m A p*) ∩ (*defer m A p*) = {}
    using *e-mod f-prof result-disj*
    by *blast*
  ultimately show *?thesis*
    by *blast*
qed

lemma *defer-not-elec-or-rej*:
  assumes
    e-mod: *electoral-module m* **and**

f-prof: *finite-profile A p*
shows *defer m A p* = *A* − (*elect m A p*) − (*reject m A p*)
proof −
from *e-mod f-prof* **have** *0*: *well-formed A* (*m A p*)
by (*simp add*: *electoral-module-def*)
hence (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
using *e-mod f-prof result-presv-alts*
by *auto*
moreover from *0* **have**
(*elect m A p*) ∩ (*defer m A p*) = {} ∧
(*reject m A p*) ∩ (*defer m A p*) = {}
using *e-mod f-prof result-disj*
by *blast*
ultimately show *?thesis*
by *blast*
qed

lemma *electoral-mod-defer-elem*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p* **and**
alternative: *x* ∈ *A* **and**
not-elected: *x* ∉ *elect m A p* **and**
not-rejected: *x* ∉ *reject m A p*
shows *x* ∈ *defer m A p*
using *DiffI e-mod f-prof alternative*
not-elected not-rejected
reject-not-elec-or-def
by *metis*

lemma *mod-contains-result-comm*:
assumes *mod-contains-result m n A p a*
shows *mod-contains-result n m A p a*
using *IntI assms electoral-mod-defer-elem empty-iff*
mod-contains-result-def result-disj
by (*smt* (*verit*, *ccfv-threshold*))

lemma *not-rej-imp-elec-or-def*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p* **and**
alternative: *x* ∈ *A* **and**
not-rejected: *x* ∉ *reject m A p*
shows *x* ∈ *elect m A p* ∨ *x* ∈ *defer m A p*
using *alternative electoral-mod-defer-elem*
e-mod not-rejected f-prof
by *metis*

lemma *single-elim-imp-red-def-set*:

assumes
eliminating: *eliminates 1 m* **and**
leftover-alternatives: *card A* > *1* **and**
f-prof: *finite-profile A p*
shows *defer m A p* ⊂ *A*
using *Diff-eq-empty-iff Diff-subset card-eq-0-iff defer-in-alts eliminates-def eliminating eq-iff leftover-alternatives not-one-le-zero f-prof psubsetI reject-not-elec-or-def*
by *metis*

lemma *eq-alts-in-profs-imp-eq-results*:
assumes
eq: ∀ *a* ∈ *A*. *prof-contains-result m A p q a* **and**

input: *electoral-module m* ∧ *finite-profile A p* ∧ *finite-profile A q*
shows *m A p* = *m A q*
proof −
have ∀ *a* ∈ *elect m A p*. *a* ∈ *elect m A q*
using *elect-in-alts eq prof-contains-result-def input in-mono*
by *metis*
moreover have ∀ *a* ∈ *elect m A q*. *a* ∈ *elect m A p*
using *contra-subsetD disjoint-iff-not-equal elect-in-alts electoral-mod-defer-elem eq prof-contains-result-def input result-disj*
by (*smt* (*verit*, *best*))
moreover have ∀ *a* ∈ *reject m A p*. *a* ∈ *reject m A q*
using *reject-in-alts eq prof-contains-result-def input in-mono*
by *fastforce*
moreover have ∀ *a* ∈ *reject m A q*. *a* ∈ *reject m A p*
using *contra-subsetD disjoint-iff-not-equal reject-in-alts electoral-mod-defer-elem eq prof-contains-result-def input result-disj*
by (*smt* (*verit*, *ccfv-SIG*))
moreover have ∀ *a* ∈ *defer m A p*. *a* ∈ *defer m A q*
using *defer-in-alts eq prof-contains-result-def input in-mono*
by *fastforce*
moreover have ∀ *a* ∈ *defer m A q*. *a* ∈ *defer m A p*
using *contra-subsetD disjoint-iff-not-equal defer-in-alts electoral-mod-defer-elem eq prof-contains-result-def input result-disj*
by (*smt* (*verit*, *best*))
ultimately show *?thesis*
using *prod.collapse subsetI subset-antisym*
by *metis*
qed

lemma *eq-def-and-elect-imp-eq*:
assumes
electoral-module m **and**

electoral-module n **and**
finite-profile A p **and**
finite-profile A q **and**
elect m A p = *elect n A q* **and**
defer m A p = *defer n A q*
shows *m A p* = *n A q*
proof −
have *disj-m*:
disjoint3 (*m A p*)
using *assms*(*1*) *assms*(*3*) *electoral-module-def*
by *auto*
have *disj-n*:
disjoint3 (*n A q*)
using *assms*(*2*) *assms*(*4*) *electoral-module-def*
by *auto*
have *set-partit-m*:
set-equals-partition A ((*elect m A p*), (*reject m A p*), (*defer m A p*))
using *assms*(*1*) *assms*(*3*) *electoral-module-def*
by *auto*
moreover have
disjoint3 ((*elect m A p*),(*reject m A p*),(*defer m A p*))
using *disj-m prod.collapse*
by *metis*
have *set-partit-n*:
set-equals-partition A ((*elect n A q*), (*reject n A q*), (*defer n A q*))
using *assms*(*2*) *assms*(*4*) *electoral-module-def*
by *auto*
moreover have
disjoint3 ((*elect n A q*),(*reject n A q*),(*defer n A q*))
using *disj-n prod.collapse*
by *metis*
have *reject-p*:
reject m A p = *A* − ((*elect m A p*) ∪ (*defer m A p*))
using *assms*(*1*) *assms*(*3*) *combine-ele-rej-def*
electoral-module-def result-imp-rej
by *metis*
have *reject-q*:
reject n A q = *A* − ((*elect n A q*) ∪ (*defer n A q*))
using *assms*(*2*) *assms*(*4*) *combine-ele-rej-def*
electoral-module-def result-imp-rej
by *metis*
from *reject-p reject-q* **show** *?thesis*
by (*simp add*: *assms*(*5*) *assms*(*6*) *prod-eqI*)
qed

2.1.5 Non-Blocking

definition *non-blocking* :: *'a Electoral-Module* ⇒ *bool* **where**
non-blocking m ≡

electoral-module m ∧
(∀ *A p*.
(($A \neq \{\}$ ∧ *finite-profile A p*) ⟶ *reject m A p* $\neq A$))

2.1.6 Electing

definition *electing* :: *'a Electoral-Module* ⇒ *bool* **where**
electing m ≡
electoral-module m ∧
(∀ *A p*. ($A \neq \{\}$ ∧ *finite-profile A p*) ⟶ *elect m A p* $\neq \{\}$)

lemma *electing-for-only-alt*:
assumes
one-alt: *card A* = *1* **and**
electing: *electing m* **and**
f-prof: *finite-profile A p*
shows *elect m A p* = *A*
using *Int-empty-right Int-insert-right card-1-singletonE*
elect-in-alts electing electing-def inf.orderE
one-alt f-prof
by (*smt* (*verit*, *del-insts*))

theorem *electing-imp-non-blocking*:
assumes *electing*: *electing m*
shows *non-blocking m*
using *Diff-disjoint Diff-empty Int-absorb2 electing*
defer-in-alts elect-in-alts electing-def
non-blocking-def reject-not-elec-or-def
by (*smt* (*verit*, *ccfv-SIG*))

2.1.7 Properties

definition *non-electing* :: *'a Electoral-Module* ⇒ *bool* **where**
non-electing m ≡
electoral-module m ∧ (∀ *A p*. *finite-profile A p* ⟶ *elect m A p* = {})

lemma *single-elim-decr-def-card*:
assumes
rejecting: *rejects 1 m* **and**
not-empty: $A \neq \{\}$ **and**
non-electing: *non-electing m* **and**
f-prof: *finite-profile A p*
shows *card* (*defer m A p*) = *card A* − *1*
using *Diff-empty One-nat-def Suc-leI card-Diff-subset card-gt-0-iff*
defer-not-elec-or-rej finite-subset non-electing
non-electing-def not-empty f-prof reject-in-alts rejecting
rejects-def
by (*smt* (*verit*, *ccfv-threshold*))

lemma *single-elim-decr-def-card2*:

assumes
eliminating: *eliminates 1 m* **and**
not-empty: *card A* $>$ *1* **and**
non-electing: *non-electing m* **and**
f-prof: *finite-profile A p*
shows *card* (*defer m A p*) = *card A* $-$ *1*
using *Diff-empty One-nat-def Suc-leI card-Diff-subset card-gt-0-iff*
defer-not-elec-or-rej finite-subset non-electing
non-electing-def not-empty f-prof reject-in-alts
eliminating eliminates-def
by (*smt* (*verit*))

definition *defer-deciding* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
defer-deciding m $\equiv$
electoral-module m $\wedge$ *non-electing m* $\wedge$ *defers 1 m*

definition *decrementing* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
decrementing m $\equiv$
electoral-module m $\wedge$ (
$\forall$ *A p* . *finite-profile A p* $\longrightarrow$
(*card A* $>$ *1* $\longrightarrow$ *card* (*reject m A p*) $\geq$ *1*))

definition *defer-condorcet-consistency* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
defer-condorcet-consistency m $\equiv$
electoral-module m $\wedge$
($\forall$ *A p w*. *condorcet-winner A p w* $\wedge$ *finite A* $\longrightarrow$
(*m A p* =
({},
A $-$ (*defer m A p*),
{*d* $\in$ *A*. *condorcet-winner A p d*})))

definition *condorcet-compatibility* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
condorcet-compatibility m $\equiv$
electoral-module m $\wedge$
($\forall$ *A p w*. *condorcet-winner A p w* $\wedge$ *finite A* $\longrightarrow$
(*w* $\notin$ *reject m A p* $\wedge$
($\forall$ *l*. $\neg$*condorcet-winner A p l* $\longrightarrow$ *l* $\notin$ *elect m A p*) $\wedge$
(*w* $\in$ *elect m A p* $\longrightarrow$
($\forall$ *l*. $\neg$*condorcet-winner A p l* $\longrightarrow$ *l* $\in$ *reject m A p*))))

definition *defer-monotonicity* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
defer-monotonicity m $\equiv$
electoral-module m $\wedge$
($\forall$ *A p q w*.
(*finite A* $\wedge$ *w* $\in$ *defer m A p* $\wedge$ *lifted A p q w*) $\longrightarrow$ *w* $\in$ *defer m A q*)

definition *defer-lift-invariance* :: *'a Electoral-Module* ⇒ *bool* **where**
defer-lift-invariance m ≡
electoral-module m ∧
(∀ *A p q a*.
($a \in$ (*defer m A p*) ∧ *lifted A p q a*) ⟶ *m A p* = *m A q*)

definition *disjoint-compatibility* :: *'a Electoral-Module* ⇒
'a Electoral-Module ⇒ *bool* **where**
disjoint-compatibility m n ≡
electoral-module m ∧ *electoral-module n* ∧
(∀ *S*. *finite S* ⟶
(∃ *A* ⊆ *S*.
(∀ *a* ∈ *A*. *indep-of-alt m S a* ∧
(∀ *p*. *finite-profile S p* ⟶ *a* ∈ *reject m S p*)) ∧
(∀ *a* ∈ *S*−*A*. *indep-of-alt n S a* ∧
(∀ *p*. *finite-profile S p* ⟶ *a* ∈ *reject n S p*))))

definition *invariant-monotonicity* :: *'a Electoral-Module* ⇒ *bool* **where**
invariant-monotonicity m ≡
electoral-module m ∧
(∀ *A p q a*. (*a* ∈ *elect m A p* ∧ *lifted A p q a*) ⟶
(*elect m A q* = *elect m A p* ∨ *elect m A q* = {*a*}))

definition *defer-invariant-monotonicity* :: *'a Electoral-Module* ⇒ *bool* **where**
defer-invariant-monotonicity m ≡
electoral-module m ∧ *non-electing m* ∧
(∀ *A p q a*. (*a* ∈ *defer m A p* ∧ *lifted A p q a*) ⟶
(*defer m A q* = *defer m A p* ∨ *defer m A q* = {*a*}))

2.1.8 Inference Rules

lemma *ccomp-and-dd-imp-def-only-winner*:
assumes *ccomp*: *condorcet-compatibility m* **and**
dd: *defer-deciding m* **and**
winner: *condorcet-winner A p w*
shows *defer m A p* = {*w*}
proof (*rule ccontr*)
assume *not-w*: *defer m A p* ≠ {*w*}
from *dd* **have** *def-1*:
defers 1 m
using *defer-deciding-def*
by *metis*
hence *c-win*:
finite-profile A p ∧ *w* ∈ *A* ∧ (∀ *x* ∈ *A* − {*w*} . *wins w p x*)
using *winner*

by *simp*
hence *card* (*defer m A p*) = *1*
using *One-nat-def Suc-leI card-gt-0-iff*
def-1 defers-def equals0D
by *metis*
hence *0*: $\exists x \in A$. *defer m A p* ={x}
using *card-1-singletonE dd defer-deciding-def*
defer-in-alts insert-subset c-win
by *metis*
with *not-w* **have** $\exists l \in A$. $l \neq w \wedge$ *defer m A p* = {l}
by *metis*
hence *not-in-defer*: $w \notin$ *defer m A p*
by *auto*
have *non-electing m*
using *dd defer-deciding-def*
by *metis*
hence *not-in-elect*: $w \notin$ *elect m A p*
using *c-win equals0D non-electing-def*
by *metis*
from *not-in-defer not-in-elect* **have** *one-side*:
$w \in$ *reject m A p*
using *ccomp condorcet-compatibility-def c-win*
electoral-mod-defer-elem
by *metis*
from *ccomp* **have** *other-side*: $w \notin$ *reject m A p*
using *condorcet-compatibility-def c-win winner*
by (*metis* (*no-types*, *hide-lams*))
thus *False*
by (*simp add*: *one-side*)
qed

theorem *ccomp-and-dd-imp-dcc*[*simp*]:
assumes *ccomp*: *condorcet-compatibility m* **and**
dd: *defer-deciding m*
shows *defer-condorcet-consistency m*
proof (*unfold defer-condorcet-consistency-def*, *auto*)
show *electoral-module m*
using *dd defer-deciding-def*
by *metis*
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a*
assume
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
finiteness: *finite A* **and**
assm: $\forall x \in A - \{w\}$.

```
        card {i. i < length p ∧ (w, x) ∈ (p!i)} <
          card {i. i < length p ∧ (x, w) ∈ (p!i)}
have winner: condorcet-winner A p w
  using assm finiteness prof-A w-in-A
  by simp
hence
  m A p =
    ({},
      A − defer m A p,
      {d ∈ A. condorcet-winner A p d})
proof −

  from dd have 0:
    elect m A p = {}
    using defer-deciding-def non-electing-def
          winner
    by fastforce

  from dd ccomp have 1: defer m A p = {w}
    using ccomp-and-dd-imp-def-only-winner winner
    by simp

  from 0 1 have 2: reject m A p = A − defer m A p
    using Diff-empty dd defer-deciding-def
          reject-not-elec-or-def winner
    by fastforce
  from 0 1 2 have 3: m A p = ({}, A − defer m A p, {w})
    using combine-ele-rej-def
    by metis
  have {w} = {d ∈ A. condorcet-winner A p d}
    using cond-winner-unique3 winner
    by metis
  thus ?thesis
    using 3
    by auto
qed
hence
  m A p =
    ({},
      A − defer m A p,
      {d ∈ A. ∀x∈A − {d}. wins d p x})
  using finiteness prof-A winner Collect-cong
  by auto
hence
  m A p =
      ({},
        A − defer m A p,
        {d ∈ A. ∀x∈A − {d}.
          prefer-count p x d < prefer-count p d x})
```

by *simp*
hence
$m\ A\ p =$
$(\{\},$
$A - \mathit{defer}\ m\ A\ p,$
$\{d \in A.\ \forall\, x{\in}A - \{d\}.$
$\mathit{card}\ \{i.\ i < \mathit{length}\ p \wedge (\mathit{let}\ r = (p!i)\ \mathit{in}\ (d \preceq_r x))\} <$
$\mathit{card}\ \{i.\ i < \mathit{length}\ p \wedge (\mathit{let}\ r = (p!i)\ \mathit{in}\ (x \preceq_r d))\}\})$
by *simp*
thus
$m\ A\ p =$
$(\{\},$
$A - \mathit{defer}\ m\ A\ p,$
$\{d \in A.\ \forall\, x{\in}A - \{d\}.$
$\mathit{card}\ \{i.\ i < \mathit{length}\ p \wedge (d,\ x) \in (p!i)\} <$
$\mathit{card}\ \{i.\ i < \mathit{length}\ p \wedge (x,\ d) \in (p!i)\}\})$
by *simp*
qed

theorem *disj-compat-comm*[*simp*]:
assumes *compatible*: *disjoint-compatibility m n*
shows *disjoint-compatibility n m*
proof −
have
$\forall\, S.\ \mathit{finite}\ S \longrightarrow$
$(\exists\, A \subseteq S.$
$(\forall\, a \in A.\ \mathit{indep\text{-}of\text{-}alt}\ n\ S\ a\ \wedge$
$(\forall\, p.\ \mathit{finite\text{-}profile}\ S\ p \longrightarrow a \in \mathit{reject}\ n\ S\ p))\ \wedge$
$(\forall\, a \in S{-}A.\ \mathit{indep\text{-}of\text{-}alt}\ m\ S\ a\ \wedge$
$(\forall\, p.\ \mathit{finite\text{-}profile}\ S\ p \longrightarrow a \in \mathit{reject}\ m\ S\ p)))$
proof
fix
$S :: {'a}\ \mathit{set}$
obtain *A* **where** *old-A*:
$\mathit{finite}\ S \longrightarrow$
$(A \subseteq S\ \wedge$
$(\forall\, a \in A.\ \mathit{indep\text{-}of\text{-}alt}\ m\ S\ a\ \wedge$
$(\forall\, p.\ \mathit{finite\text{-}profile}\ S\ p \longrightarrow a \in \mathit{reject}\ m\ S\ p))\ \wedge$
$(\forall\, a \in S{-}A.\ \mathit{indep\text{-}of\text{-}alt}\ n\ S\ a\ \wedge$
$(\forall\, p.\ \mathit{finite\text{-}profile}\ S\ p \longrightarrow a \in \mathit{reject}\ n\ S\ p)))$
using *compatible disjoint-compatibility-def*
by *fastforce*
hence
$\mathit{finite}\ S \longrightarrow$
$(\exists\, A \subseteq S.$
$(\forall\, a \in S{-}A.\ \mathit{indep\text{-}of\text{-}alt}\ n\ S\ a\ \wedge$
$(\forall\, p.\ \mathit{finite\text{-}profile}\ S\ p \longrightarrow a \in \mathit{reject}\ n\ S\ p))\ \wedge$
$(\forall\, a \in A.\ \mathit{indep\text{-}of\text{-}alt}\ m\ S\ a\ \wedge$

$(\forall\, p.\ \textit{finite-profile}\ S\ p \longrightarrow a \in \textit{reject}\ m\ S\ p)))$
by *auto*
hence
finite $S \longrightarrow$
$(\exists\, A \subseteq S.$
$(\forall\, a \in S{-}A.\ \textit{indep-of-alt}\ n\ S\ a\ \wedge$
$(\forall\, p.\ \textit{finite-profile}\ S\ p \longrightarrow a \in \textit{reject}\ n\ S\ p))\ \wedge$
$(\forall\, a \in S{-}(S{-}A).\ \textit{indep-of-alt}\ m\ S\ a\ \wedge$
$(\forall\, p.\ \textit{finite-profile}\ S\ p \longrightarrow a \in \textit{reject}\ m\ S\ p)))$
using *double-diff order-refl*
by *metis*
thus
finite $S \longrightarrow$
$(\exists\, A \subseteq S.$
$(\forall\, a \in A.\ \textit{indep-of-alt}\ n\ S\ a\ \wedge$
$(\forall\, p.\ \textit{finite-profile}\ S\ p \longrightarrow a \in \textit{reject}\ n\ S\ p))\ \wedge$
$(\forall\, a \in S{-}A.\ \textit{indep-of-alt}\ m\ S\ a\ \wedge$
$(\forall\, p.\ \textit{finite-profile}\ S\ p \longrightarrow a \in \textit{reject}\ m\ S\ p)))$
by *fastforce*
qed
moreover have *electoral-module* $m\ \wedge$ *electoral-module* n
using *compatible disjoint-compatibility-def*
by *auto*
ultimately show *?thesis*
by (*simp add*: *disjoint-compatibility-def*)
qed

theorem *dl-inv-imp-def-mono*[*simp*]:
assumes *defer-lift-invariance* m
shows *defer-monotonicity* m
using *assms defer-monotonicity-def defer-lift-invariance-def*
by *fastforce*

2.1.9 Social Choice Properties

Condorcet Consistency

definition *condorcet-consistency* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
condorcet-consistency $m \equiv$
electoral-module $m\ \wedge$
$(\forall\ A\ p\ w.\ \textit{condorcet-winner}\ A\ p\ w \longrightarrow$
$(m\ A\ p =$
$(\{e \in A.\ \textit{condorcet-winner}\ A\ p\ e\},$
$A - (\textit{elect}\ m\ A\ p),$
$\{\})))$

lemma *condorcet-consistency2*:
condorcet-consistency $m \longleftrightarrow$
electoral-module $m\ \wedge$

```
    (∀ A p w. condorcet-winner A p w ⟶
        (m A p =
          ({w}, A − (elect m A p), {})))
proof (auto)
  show condorcet-consistency m ⟹ electoral-module m
    using condorcet-consistency-def
    by metis
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a
  assume
    cc: condorcet-consistency m
  have assm0:
    condorcet-winner A p w ⟹ m A p = ({w}, A − elect m A p, {})
    using cond-winner-unique3 condorcet-consistency-def cc
    by (metis (mono-tags, lifting))
  assume
    finite-A: finite A and
    prof-A: profile A p and
    w-in-A: w ∈ A
  also have
    ∀ x∈A − {w}.
      prefer-count p w x > prefer-count p x w ⟹
        condorcet-winner A p w
    using finite-A prof-A w-in-A wins.elims
    by simp
  ultimately show
    ∀ x∈A − {w}.
        card {i. i < length p ∧ (w, x) ∈ (p!i)} <
            card {i. i < length p ∧ (x, w) ∈ (p!i)} ⟹
                m A p = ({w}, A − elect m A p, {})
    using assm0
    by auto
next
  have assm0:
    electoral-module m ⟹
      ∀ A p w. condorcet-winner A p w ⟶
          m A p = ({w}, A − elect m A p, {}) ⟹
            condorcet-consistency m
    using condorcet-consistency-def cond-winner-unique3
    by (smt (verit, del-insts))
  assume e-mod:
    electoral-module m
  thus
    ∀ A p w. finite A ∧ profile A p ∧ w ∈ A ∧
       (∀ x∈A − {w}.
          card {i. i < length p ∧ (w, x) ∈ (p!i)} <
```

$card\ \{i.\ i < length\ p \wedge (x,\ w) \in (p!i)\}) \longrightarrow$
$m\ A\ p = (\{w\},\ A - elect\ m\ A\ p,\ \{\}) \Longrightarrow$
condorcet-consistency m
using *assm0 e-mod*
by *simp*
qed

(Weak) Monotonicity

definition *monotonicity* :: *'a Electoral-Module* ⇒ *bool* **where**
monotonicity m ≡
electoral-module m ∧
$(\forall A\ p\ q\ w.$
$(finite\ A \wedge w \in elect\ m\ A\ p \wedge lifted\ A\ p\ q\ w) \longrightarrow w \in elect\ m\ A\ q)$

Homogeneity

fun *times* :: *nat* ⇒ *'a list* ⇒ *'a list* **where**
times n l = *concat* (*replicate n l*)

definition *homogeneity* :: *'a Electoral-Module* ⇒ *bool* **where**
homogeneity m ≡
electoral-module m ∧
$(\forall\ A\ p\ n\ .$
$(finite\text{-}profile\ A\ p \wedge n > 0 \longrightarrow$
$(m\ A\ p = m\ A\ (times\ n\ p))))$

end

2.2 Evaluation Function

theory *Evaluation-Function*
imports *Social-Choice-Types/Profile*
begin

This is the evaluation function. From a set of currently eligible alternatives, the evaluation function computes a numerical value that is then to be used for further (s)election, e.g., by the elimination module.

2.2.1 Definition

type-synonym *'a Evaluation-Function* = *'a* ⇒ *'a set* ⇒ *'a Profile* ⇒ *nat*

2.2.2 Property

definition *condorcet-rating* :: *'a Evaluation-Function* ⇒ *bool* **where**

condorcet-rating f ≡
  ∀ *A p w* . *condorcet-winner A p w* ⟶
    (∀ *l* ∈ *A* . *l* ≠ *w* ⟶ *f l A p* < *f w A p*)

2.2.3 Theorems

theorem *cond-winner-imp-max-eval-val*:
  assumes
    rating: *condorcet-rating e* **and**
    f-prof: *finite-profile A p* **and**
    winner: *condorcet-winner A p w*
  shows *e w A p* = *Max* {*e a A p* | *a*. *a* ∈ *A*}
proof −

  let *?set* = {*e a A p* | *a*. *a* ∈ *A*} **and**
      ?eMax = *Max* {*e a A p* | *a*. *a* ∈ *A*} **and**
      ?eW = *e w A p*

  from *f-prof* **have** *0*: *finite ?set*
    by *simp*

  have *1*: *?set* ≠ {}
    using *condorcet-winner.simps winner*
    by *fastforce*

  have *2*: *?eW* ∈ *?set*
    using *CollectI condorcet-winner.simps winner*
    by (*metis* (*mono-tags*, *lifting*))

  have *3*: ∀ *e* ∈ *?set* . *e* ≤ *?eW*
    using *CollectD condorcet-rating-def eq-iff*
        order.strict-implies-order rating winner
    by (*smt* (*verit*, *best*))

  from *2 3* **have** *4*:
    ?eW ∈ *?set* ∧ (∀ *a* ∈ *?set*. *a* ≤ *?eW*)
    by *blast*
  from *0 1 4 Max-eq-iff* **show** *?thesis*
    by (*metis* (*no-types*, *lifting*))
qed

theorem *non-cond-winner-not-max-eval*:
  assumes
    rating: *condorcet-rating e* **and**
    f-prof: *finite-profile A p* **and**
    winner: *condorcet-winner A p w* **and**
    linA: *l* ∈ *A* **and**
    loser: *w* ≠ *l*

shows $e\ l\ A\ p < Max\ \{e\ a\ A\ p \mid a.\ a \in A\}$
proof −
have $e\ l\ A\ p < e\ w\ A\ p$
using *condorcet-rating-def linA loser rating winner*
by *metis*
also have $e\ w\ A\ p = Max\ \{e\ a\ A\ p \mid a.\ a \in A\}$
using *cond-winner-imp-max-eval-val f-prof rating winner*
by *fastforce*
finally show *?thesis*
by *simp*
qed

end

2.3 Elimination Module

theory *Elimination-Module*
imports *Evaluation-Function*
Electoral-Module
begin

This is the elimination module. It rejects a set of alternatives only if these are not all alternatives. The alternatives potentially to be rejected are put in a so-called elimination set. These are all alternatives that score below a preset threshold value that depends on the specific voting rule.

2.3.1 Definition

type-synonym *Threshold-Value* = *nat*

type-synonym *'a Electoral-Set* = *'a set* ⇒ *'a Profile* ⇒ *'a set*

fun *elimination-set* :: *'a Evaluation-Function* ⇒ *Threshold-Value* ⇒
(*nat* ⇒ *Threshold-Value* ⇒ *bool*) ⇒
'a Electoral-Set **where**
elimination-set $e\ t\ r\ A\ p = \{a \in A\ .\ r\ (e\ a\ A\ p)\ t\ \}$

fun *elimination-module* :: *'a Evaluation-Function* ⇒ *Threshold-Value* ⇒
(*nat* ⇒ *nat* ⇒ *bool*) ⇒ *'a Electoral-Module* **where**
elimination-module $e\ t\ r\ A\ p =$
(*if* (*elimination-set* $e\ t\ r\ A\ p$) $\neq A$
then ({}, (*elimination-set* $e\ t\ r\ A\ p$), A − (*elimination-set* $e\ t\ r\ A\ p$))
else ({},{},A))

2.3.2 Common Eliminators

fun *less-eliminator* :: *'a Evaluation-Function* ⇒ *Threshold-Value* ⇒
'a Electoral-Module **where**
less-eliminator e t A p = elimination-module e t (<) *A p*

fun *max-eliminator* :: *'a Evaluation-Function* ⇒ *'a Electoral-Module* **where**
max-eliminator e A p =
less-eliminator e (*Max* {*e x A p* | *x. x* ∈ *A*}) *A p*

fun *leq-eliminator* :: *'a Evaluation-Function* ⇒ *Threshold-Value* ⇒
'a Electoral-Module **where**
leq-eliminator e t A p = elimination-module e t (≤) *A p*

fun *min-eliminator* :: *'a Evaluation-Function* ⇒ *'a Electoral-Module* **where**
min-eliminator e A p =
leq-eliminator e (*Min* {*e x A p* | *x. x* ∈ *A*}) *A p*

fun *average* :: *'a Evaluation-Function* ⇒ *'a set* ⇒ *'a Profile* ⇒
Threshold-Value **where**
average e A p = ($\sum$ *x* ∈ *A. e x A p*) *div* (*card A*)

fun *less-average-eliminator* :: *'a Evaluation-Function* ⇒
'a Electoral-Module **where**
less-average-eliminator e A p = less-eliminator e (*average e A p*) *A p*

fun *leq-average-eliminator* :: *'a Evaluation-Function* ⇒
'a Electoral-Module **where**
leq-average-eliminator e A p = leq-eliminator e (*average e A p*) *A p*

2.3.3 Soundness

lemma *elim-mod-sound*[*simp*]: *electoral-module* (*elimination-module e t r*)
proof (*unfold electoral-module-def*, *safe*)
fix
A :: *'a set* **and**
p :: *'a Profile*
have *set-equals-partition A* (*elimination-module e t r A p*)
by *auto*
thus *well-formed A* (*elimination-module e t r A p*)
by *simp*
qed

lemma *less-elim-sound*[*simp*]: *electoral-module* (*less-eliminator e t*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show

$\{a \in A.\ e\ a\ A\ p < t\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p < t\} \cup A = A$
by *safe*
qed

lemma *leq-elim-sound*[*simp*]: *electoral-module* (*leq-eliminator e t*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p \leq t\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p \leq t\} \cup A = A$
by *safe*
qed

lemma *max-elim-sound*[*simp*]: *electoral-module* (*max-eliminator e*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p < Max\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p < Max\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \cup A = A$
by *safe*
qed

lemma *min-elim-sound*[*simp*]: *electoral-module* (*min-eliminator e*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p \leq Min\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p \leq Min\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \cup A = A$
by *safe*
qed

lemma *less-avg-elim-sound*[*simp*]: *electoral-module* (*less-average-eliminator e*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p < (\sum x{\in}A.\ e\ x\ A\ p)\ div\ card\ A\} \neq A \longrightarrow$

$\{a \in A.\ e\ a\ A\ p < (\sum x{\in}A.\ e\ x\ A\ p)\ div\ card\ A\} \cup A = A$
by *safe*
qed

lemma *leq-avg-elim-sound*[*simp*]: *electoral-module* (*leq-average-eliminator e*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p \leq (\sum x{\in}A.\ e\ x\ A\ p)\ div\ card\ A\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p \leq (\sum x{\in}A.\ e\ x\ A\ p)\ div\ card\ A\} \cup A = A$
by *safe*
qed

2.3.4 Non-Electing

lemma *elim-mod-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*elimination-module e t r*)
by (*simp add*: *non-electing-def*)

lemma *less-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*less-eliminator e t*)
using *elim-mod-non-electing profile less-elim-sound*
by (*simp add*: *non-electing-def*)

lemma *leq-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*leq-eliminator e t*)
proof −
have *non-electing* (*elimination-module e t* ($\leq$))
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

lemma *max-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*max-eliminator e*)
proof −
have *non-electing* (*elimination-module e t* ($<$))
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

lemma *min-elim-non-electing*:
assumes *profile*: *finite-profile* A p
shows *non-electing* (*min-eliminator* e)
proof −
have *non-electing* (*elimination-module* e t $(<)$)
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

lemma *less-avg-elim-non-electing*:
assumes *profile*: *finite-profile* A p
shows *non-electing* (*less-average-eliminator* e)
proof −
have *non-electing* (*elimination-module* e t $(<)$)
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

lemma *leq-avg-elim-non-electing*:
assumes *profile*: *finite-profile* A p
shows *non-electing* (*leq-average-eliminator* e)
proof −
have *non-electing* (*elimination-module* e t $(\leq)$)
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

2.3.5 Inference Rules

theorem *cr-eval-imp-ccomp-max-elim*[*simp*]:
assumes
profile: *finite-profile* A p **and**
rating: *condorcet-rating* e
shows
condorcet-compatibility (*max-eliminator* e)
unfolding *condorcet-compatibility-def*
proof (*auto*)
have *f1*:
$\bigwedge A\ p\ w\ x.\ \textit{condorcet-winner}\ A\ p\ w \Longrightarrow$
$\textit{finite}\ A \Longrightarrow w \in A \Longrightarrow e\ w\ A\ p < \textit{Max}\ \{e\ x\ A\ p\ |x.\ x \in A\} \Longrightarrow$
$x \in A \Longrightarrow e\ x\ A\ p < \textit{Max}\ \{e\ x\ A\ p\ |x.\ x \in A\}$
using *rating*
by (*simp add*: *cond-winner-imp-max-eval-val*)
thus
$\bigwedge A\ p\ w\ x.$
$\textit{profile}\ A\ p \Longrightarrow w \in A \Longrightarrow$

```
        ∀ x∈A − {w}.
          card {i. i < length p ∧ (w, x) ∈ (p!i)} <
            card {i. i < length p ∧ (x, w) ∈ (p!i)} ⟹
              finite A ⟹ e w A p < Max {e x A p | x. x ∈ A} ⟹
                x ∈ A ⟹ e x A p < Max {e x A p | x. x ∈ A}
    by simp
qed

lemma cr-eval-imp-dcc-max-elim-helper1:
  assumes
    f-prof: finite-profile A p and
    rating: condorcet-rating e and
    winner: condorcet-winner A p w
  shows elimination-set e (Max {e x A p | x. x ∈ A}) (<) A p = A − {w}
proof (safe, simp-all, safe)
  assume
    w-in-A: w ∈ A and
    max: e w A p < Max {e x A p |x. x ∈ A}
  show False
    using cond-winner-imp-max-eval-val
          rating winner f-prof max
    by fastforce
next
  fix
    x :: 'a
  assume
    x-in-A: x ∈ A and
    not-max: ¬ e x A p < Max {e y A p |y. y ∈ A}
  show x = w
    using non-cond-winner-not-max-eval x-in-A
          rating winner f-prof not-max
    by (metis (mono-tags, lifting))
qed


theorem cr-eval-imp-dcc-max-elim[simp]:
  assumes rating: condorcet-rating e
  shows defer-condorcet-consistency (max-eliminator e)
  unfolding defer-condorcet-consistency-def
proof (safe, simp)
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a
  assume
    winner: condorcet-winner A p w and
    finite: finite A
  let ?trsh = (Max {e y A p | y. y ∈ A})
  show
```

```
  max-eliminator e A p =
    ({},
      A − defer (max-eliminator e) A p,
      {a ∈ A. condorcet-winner A p a})
proof (cases elimination-set e (?trsh) (<) A p ≠ A)
  case True
  have profile: finite-profile A p
    using winner
    by simp
  with rating winner have 0:
    (elimination-set e ?trsh (<) A p) = A − {w}
    using cr-eval-imp-dcc-max-elim-helper1
    by (metis (mono-tags, lifting))
  have
    max-eliminator e A p =
      ({},
        (elimination-set e ?trsh (<) A p),
        A − (elimination-set e ?trsh (<) A p))
    using True
    by simp
  also have ... = ({}, A − {w}, A − (A − {w}))
    using 0
    by presburger
  also have ... = ({}, A − {w}, {w})
    using winner
    by auto
  also have ... = ({},A − defer (max-eliminator e) A p, {w})
    using calculation
    by auto
  also have
    ... =
      ({},
        A − defer (max-eliminator e) A p,
        {d ∈ A. condorcet-winner A p d})
    using cond-winner-unique3 winner Collect-cong
    by (metis (no-types, lifting))
  finally show ?thesis
    using finite winner
    by metis
next
  case False
  thus ?thesis
  proof −
    have f1:
      finite A ∧ profile A p ∧ w ∈ A ∧ (∀ a. a ∉ A − {w} ∨ wins w p a)
      using winner
      by auto
    hence
      ?trsh = e w A p
```

```
      using rating winner
      by (simp add: cond-winner-imp-max-eval-val)
    hence False
      using f1 False
      by auto
    thus ?thesis
      by simp
  qed
 qed
qed

end
```

2.4 Aggregator

```
theory Aggregator
  imports Social-Choice-Types/Result
begin
```

An aggregator gets two partitions (results of electoral modules) as input and output another partition. They are used to aggregate results of parallel composed electoral modules. They are commutative, i.e., the order of the aggregated modules does not affect the resulting aggregation. Moreover, they are conservative in the sense that the resulting decisions are subsets of the two given partitions' decisions.

2.4.1 Definition

```
type-synonym 'a Aggregator = 'a set ⇒ 'a Result ⇒ 'a Result ⇒ 'a Result

definition aggregator :: 'a Aggregator ⇒ bool where
  aggregator agg ≡
   ∀ A e1 e2 d1 d2 r1 r2.
    (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
    well-formed A (agg A (e1, r1, d1) (e2, r2, d2))
```

2.4.2 Properties

```
definition agg-commutative :: 'a Aggregator ⇒ bool where
  agg-commutative agg ≡
   aggregator agg ∧ (∀ A e1 e2 d1 d2 r1 r2.
    agg A (e1, r1, d1) (e2, r2, d2) = agg A (e2, r2, d2) (e1, r1, d1))
```

```
definition agg-conservative :: 'a Aggregator ⇒ bool where
  agg-conservative agg ≡
    aggregator agg ∧
    (∀ A e1 e2 d1 d2 r1 r2.
      ((well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
        elect-r (agg A (e1, r1, d1) (e2, r2, d2)) ⊆ (e1 ∪ e2) ∧
        reject-r (agg A (e1, r1, d1) (e2, r2, d2)) ⊆ (r1 ∪ r2) ∧
        defer-r (agg A (e1, r1, d1) (e2, r2, d2)) ⊆ (d1 ∪ d2)))

end
```

2.5 Maximum Aggregator

```
theory Maximum-Aggregator
  imports Aggregator
begin
```

The max(imum) aggregator takes two partitions of an alternative set A as input. It returns a partition where every alternative receives the maximum result of the two input partitions.

2.5.1 Definition

```
fun max-aggregator :: 'a Aggregator where
  max-aggregator A (e1, r1, d1) (e2, r2, d2) =
    (e1 ∪ e2,
     A − (e1 ∪ e2 ∪ d1 ∪ d2),
     (d1 ∪ d2) − (e1 ∪ e2))
```

2.5.2 Auxiliary Lemma

```
lemma max-agg-rej-set: (well-formed A (e1, r1, d1) ∧
                          well-formed A (e2, r2, d2)) ⟶
          reject-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) = r1 ∩ r2
proof −
  have well-formed A (e1, r1, d1) ⟶ A − (e1 ∪ d1) = r1
    by (simp add: result-imp-rej)
  moreover have
    well-formed A (e2, r2, d2) ⟶ A − (e2 ∪ d2) = r2
    by (simp add: result-imp-rej)
  ultimately have
    (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
        A − (e1 ∪ e2 ∪ d1 ∪ d2) = r1 ∩ r2
    by blast
  moreover have
```

$\{l \in A.\ l \notin e1 \cup e2 \cup d1 \cup d2\} = A - (e1 \cup e2 \cup d1 \cup d2)$
    by (*simp add*: *set-diff-eq*)
  ultimately show *?thesis*
    by *simp*
qed

2.5.3 Soundness

theorem *max-agg-sound*[*simp*]: *aggregator max-aggregator*
  unfolding *aggregator-def*
proof (*simp*, *safe*)
  fix
    $A :: 'a\ set$ **and**
    $e1 :: 'a\ set$ **and**
    $e2 :: 'a\ set$ **and**
    $d1 :: 'a\ set$ **and**
    $d2 :: 'a\ set$ **and**
    $r1 :: 'a\ set$ **and**
    $r2 :: 'a\ set$ **and**
    $x :: 'a$
  assume
    asm1: $e2 \cup r2 \cup d2 = e1 \cup r1 \cup d1$ **and**
    asm2: $x \notin d1$ **and**
    asm3: $x \notin r1$ **and**
    asm4: $x \in e2$
  show $x \in e1$
    using *asm1 asm2 asm3 asm4*
    by *auto*
next
  fix
    $A :: 'a\ set$ **and**
    $e1 :: 'a\ set$ **and**
    $e2 :: 'a\ set$ **and**
    $d1 :: 'a\ set$ **and**
    $d2 :: 'a\ set$ **and**
    $r1 :: 'a\ set$ **and**
    $r2 :: 'a\ set$ **and**
    $x :: 'a$
  assume
    asm1: $e2 \cup r2 \cup d2 = e1 \cup r1 \cup d1$ **and**
    asm2: $x \notin d1$ **and**
    asm3: $x \notin r1$ **and**
    asm4: $x \in d2$
  show $x \in e1$
    using *asm1 asm2 asm3 asm4*
    by *auto*
qed

2.5.4 Properties

theorem *max-agg-consv*[*simp*]: *agg-conservative max-aggregator*
proof −
  have
    $\forall$ *A e1 e2 d1 d2 r1 r2*.
        (*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
      reject-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) = *r1* $\cap$ *r2*
    using *max-agg-rej-set*
    by *blast*
  hence
    $\forall$ *A e1 e2 d1 d2 r1 r2*.
          (*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
        reject-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ *r1* $\cap$ *r2*
    by *blast*
  moreover have
    $\forall$ *A e1 e2 d1 d2 r1 r2*.
      (*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
        elect-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*e1* $\cup$ *e2*)
    by (*simp add*: *subset-eq*)
  ultimately have
    $\forall$ *A e1 e2 d1 d2 r1 r2*.
      (*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
        (*elect-r* (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*e1* $\cup$ *e2*) $\wedge$
         reject-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*r1* $\cup$ *r2*))
    by *blast*
  moreover have
    $\forall$ *A e1 e2 d1 d2 r1 r2*.
      (*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
        defer-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*d1* $\cup$ *d2*)
    by *auto*
  ultimately have
    $\forall$ *A e1 e2 d1 d2 r1 r2*.
      (*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
        (*elect-r* (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*e1* $\cup$ *e2*) $\wedge$
        reject-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*r1* $\cup$ *r2*) $\wedge$
        defer-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) $\subseteq$ (*d1* $\cup$ *d2*))
    by *blast*
  thus *?thesis*
    by (*simp add*: *agg-conservative-def*)
qed

theorem *max-agg-comm*[*simp*]: *agg-commutative max-aggregator*
  unfolding *agg-commutative-def*
proof (*safe*)
  show *aggregator max-aggregator*
    by *simp*
next
  fix

A :: *'a set* **and**
e1 :: *'a set* **and**
e2 :: *'a set* **and**
d1 :: *'a set* **and**
d2 :: *'a set* **and**
r1 :: *'a set* **and**
r2 :: *'a set*
show
max-aggregator A (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*) =
max-aggregator A (*e2*, *r2*, *d2*) (*e1*, *r1*, *d1*)
by *auto*
qed

end

2.6 Termination Condition

theory *Termination-Condition*
imports *Social-Choice-Types/Result*
begin

The termination condition is used in loops. It decides whether or not to terminate the loop after each iteration, depending on the current state of the loop.

2.6.1 Definition

type-synonym *'a Termination-Condition* = *'a Result* ⇒ *bool*

end

2.7 Defer Equal Condition

theory *Defer-Equal-Condition*
imports *Termination-Condition*
begin

This is a family of termination conditions. For a natural number n, the according defer-equal condition is true if and only if the given result's defer-set contains exactly n elements.

2.7.1 Definition

fun *defer-equal-condition* :: *nat* ⇒ *'a Termination-Condition* **where**
defer-equal-condition n result = (*let* (*e*, *r*, *d*) = *result in card d* = *n*)

end

Chapter 3

Basic Modules

3.1 Defer Module

theory *Defer-Module*
  imports *Component-Types/Electoral-Module*
begin

The defer module is not concerned about the voter's ballots, and simply defers all alternatives. It is primarily used for defining an empty loop.

3.1.1 Definition

fun *defer-module* :: *'a Electoral-Module* **where**
  defer-module A p = ({}, {}, *A*)

3.1.2 Soundness

theorem *def-mod-sound*[*simp*]: *electoral-module defer-module*
  unfolding *electoral-module-def*
  by *simp*

3.1.3 Properties

theorem *def-mod-non-electing*: *non-electing defer-module*
  unfolding *non-electing-def*
  by *simp*

theorem *def-mod-def-lift-inv*: *defer-lift-invariance defer-module*
  unfolding *defer-lift-invariance-def*
  by *simp*

end

3.2 Drop Module

theory *Drop-Module*
  imports *Component-Types/Electoral-Module*
begin

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according drop module rejects the lexicographically first n alternatives (from A) and defers the rest. It is primarily used as counterpart to the pass module in a parallel composition, in order to segment the alternatives into two groups.

3.2.1 Definition

fun *drop-module* :: $nat \Rightarrow {'a}\ Preference\text{-}Relation \Rightarrow {'a}\ Electoral\text{-}Module$ **where**
  $drop\text{-}module\ n\ r\ A\ p =$
    $(\{\},$
    $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\},$
    $\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\})$

3.2.2 Soundness

theorem *drop-mod-sound*[*simp*]:
  assumes *order*: *linear-order r*
  shows *electoral-module* (*drop-module n r*)
proof −
  let $?mod = drop\text{-}module\ n\ r$
  have
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $(\forall a \in A.\ a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\} \vee$
        $a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\})$
    by *auto*
  hence
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} \cup$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} = A$
    by *blast*
  hence *0*:
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $set\text{-}equals\text{-}partition\ A\ (drop\text{-}module\ n\ r\ A\ p)$
    by *simp*
  have
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $(\forall a \in A.\ \neg(a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\} \wedge$
        $a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\}))$
    by *auto*
  hence
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} \cap$

$\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} = \{\}$
by *blast*
hence *1*: $\forall A\ p.$ *finite-profile* $A\ p \longrightarrow$ *disjoint3* (*?mod* $A\ p$)
by *simp*
from *0 1* **have**
$\forall A\ p.$ *finite-profile* $A\ p \longrightarrow$
well-formed A (*?mod* $A\ p$)
by *simp*
hence
$\forall A\ p.$ *finite-profile* $A\ p \longrightarrow$
well-formed A (*?mod* $A\ p$)
by *simp*
thus *?thesis*
using *electoral-modI*
by *metis*
qed

3.2.3 Non-Electing

theorem *drop-mod-non-electing*[*simp*]:
assumes *order*: *linear-order r*
shows *non-electing* (*drop-module n r*)
by (*simp add*: *non-electing-def order*)

3.2.4 Properties

theorem *drop-mod-def-lift-inv*[*simp*]:
assumes *order*: *linear-order r*
shows *defer-lift-invariance* (*drop-module n r*)
by (*simp add*: *order defer-lift-invariance-def*)

end

3.3 Pass Module

theory *Pass-Module*
imports *Component-Types/Electoral-Module*
begin

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according pass module defers the lexicographically first n alternatives (from A) and rejects the rest. It is primarily used as counterpart to the drop module in a parallel composition in order to segment the alternatives into two groups.

3.3.1 Definition

fun *pass-module* :: *nat* ⇒ *'a Preference-Relation* ⇒ *'a Electoral-Module* **where**
  pass-module n r A p =
    ({},
    $\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\}$,
    $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\}$)

3.3.2 Soundness

theorem *pass-mod-sound*[*simp*]:
  assumes *order*: *linear-order r*
  shows *electoral-module* (*pass-module n r*)
proof –
  let *?mod* = *pass-module n r*
  have
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
        $(\forall a \in A.\ a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\} \lor$
            $a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\})$
    using *CollectI not-less*
    by *metis*
  hence
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
        $\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} \cup$
        $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} = A$
    by *blast*
  hence *0*:
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$ *set-equals-partition A* (*pass-module n r A p*)
    by *simp*
  have
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $(\forall a \in A.\ \neg(a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\} \land$
          $a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\}))$
    by *auto*
  hence
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} \cap$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} = \{\}$
    by *blast*
  hence *1*:
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$ *disjoint3* (*?mod A p*)
    by *simp*
  from *0 1*
  have
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$ *well-formed A* (*?mod A p*)
    by *simp*
  hence
    $\forall\ A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$ *well-formed A* (*?mod A p*)
    by *simp*
  thus *?thesis*

```
    using electoral-modI
    by metis
qed
```

3.3.3 Non-Blocking

```
theorem pass-mod-non-blocking[simp]:
  assumes order: linear-order r and
          g0-n: n > 0
  shows non-blocking (pass-module n r)
proof −
  have ∀ A p. (A ≠ {} ∧ finite-profile A p) ⟶
          reject (pass-module n r) A p ≠ A
  proof
    fix
      A :: 'a set
    show
      ∀ p. (A ≠ {} ∧ finite-profile A p) ⟶
          reject (pass-module n r) A p ≠ A
    proof
      fix
        p :: 'a Profile
      show
        (A ≠ {} ∧ finite-profile A p) ⟶
          reject (pass-module n r) A p ≠ A
      proof
        assume input-fine: A ≠ {} ∧ finite-profile A p
        hence finite A
          by simp
        moreover have A ≠ {}
          by (simp add: input-fine)
        moreover have linear-order-on A (limit A r)
          using limit-presv-lin-ord order
          by auto
        ultimately have
          ∃ a∈A. above (limit A r) a = {a} ∧
                (∀ x∈A. above (limit A r) x = {x} ⟶ x = a)
          by (simp add: above-one)
        hence {a ∈ A. card(above (limit A r) a) > n} ≠ A
          using One-nat-def Suc-leI assms(2) is-singletonI
                is-singleton-altdef leD mem-Collect-eq
          by (metis (no-types, lifting))
        thus reject (pass-module n r) A p ≠ A
          by simp
      qed
    qed
  qed
  thus ?thesis
    by (simp add: non-blocking-def order)
```

qed

3.3.4 Non-Electing

theorem *pass-mod-non-electing*[*simp*]:
  assumes *order*: *linear-order r*
  shows *non-electing* (*pass-module n r*)
  by (*simp add*: *non-electing-def order*)

3.3.5 Properties

theorem *pass-mod-dl-inv*[*simp*]:
  assumes *order*: *linear-order r*
  shows *defer-lift-invariance* (*pass-module n r*)
  by (*simp add*: *order defer-lift-invariance-def*)

theorem *pass-one-mod-def-one*[*simp*]:
  assumes *order*: *linear-order r*
  shows *defers 1* (*pass-module 1 r*)
proof –
  have $\forall$ *A p*. (*card A* $\geq$ *1* $\wedge$ *finite-profile A p*) $\longrightarrow$
        card (*defer* (*pass-module 1 r*) *A p*) = *1*
  proof
    fix
      A :: *'a set*
    show
      $\forall$ *p*. (*card A* $\geq$ *1* $\wedge$ *finite-profile A p*) $\longrightarrow$
        card (*defer* (*pass-module 1 r*) *A p*) = *1*
    proof
      fix
        p :: *'a Profile*
      show
        (*card A* $\geq$ *1* $\wedge$ *finite-profile A p*) $\longrightarrow$
          card (*defer* (*pass-module 1 r*) *A p*) = *1*
      proof
        assume
          A-valid: (*card A* $\geq$ *1* $\wedge$ *finite-profile A p*)
        hence *finite-A*: *finite A*
          by *simp*
        moreover have *A* $\neq$ {}
          using *A-valid*
          by *auto*
        moreover have *lin-ord-on-A*:
          linear-order-on A (*limit A r*)
          using *order limit-presv-lin-ord*
          by *blast*
        ultimately have *winner-exists*:
          $\exists$ *a*$\in$*A*. *above* (*limit A r*) *a* = {*a*} $\wedge$
            ($\forall$ *x*$\in$*A*. *above* (*limit A r*) *x* = {*x*} $\longrightarrow$ *x* = *a*)

by (*simp add*: *above-one*)
then obtain *w* **where** *w-unique-top*:
above (*limit A r*) *w* = {*w*} ∧
(∀ *x*∈*A*. *above* (*limit A r*) *x* = {*x*} ⟶ *x* = *w*)
using *above-one*
by *auto*
hence {*a* ∈ *A*. *card*(*above* (*limit A r*) *a*) ≤ *1*} = {*w*}
proof
assume
w-top: *above* (*limit A r*) *w* = {*w*} **and**
w-unique: ∀ *x*∈*A*. *above* (*limit A r*) *x* = {*x*} ⟶ *x* = *w*
have *card* (*above* (*limit A r*) *w*) ≤ *1*
using *w-top*
by *auto*
hence {*w*} ⊆ {*a* ∈ *A*. *card*(*above* (*limit A r*) *a*) ≤ *1*}
using *winner-exists w-unique-top*
by *blast*
moreover have
{*a* ∈ *A*. *card*(*above* (*limit A r*) *a*) ≤ *1*} ⊆ {*w*}
proof
fix
x :: '*a*
assume *x-in-winner-set*:
x ∈ {*a* ∈ *A*. *card* (*above* (*limit A r*) *a*) ≤ *1*}
hence *x-in-A*: *x* ∈ *A*
by *auto*
hence *connex-limit*:
connex A (*limit A r*)
using *lin-ord-imp-connex lin-ord-on-A*
by *simp*
hence *let q* = *limit A r in x* $\preceq_q$ *x*
using *connex-limit above-connex*
pref-imp-in-above x-in-A
by *metis*
hence (*x*,*x*) ∈ *limit A r*
by *simp*
hence *x-above-x*: *x* ∈ *above* (*limit A r*) *x*
by (*simp add*: *above-def*)
have *above* (*limit A r*) *x* ⊆ *A*
using *above-presv-limit order*
by *fastforce*
hence *above-finite*: *finite* (*above* (*limit A r*) *x*)
by (*simp add*: *A-valid finite-subset*)
have *card* (*above* (*limit A r*) *x*) ≤ *1*
using *x-in-winner-set*
by *simp*
moreover have
card (*above* (*limit A r*) *x*) ≥ *1*
using *One-nat-def Suc-leI above-finite card-eq-0-iff*

```
            equals0D neq0-conv x-above-x
          by metis
        ultimately have
          card (above (limit A r) x) = 1
          by simp
        hence {x} = above (limit A r) x
          using is-singletonE is-singleton-altdef singletonD x-above-x
          by metis
        hence x = w
          using w-unique
          by (simp add: x-in-A)
        thus x ∈ {w}
          by simp
      qed
      ultimately have
        {w} = {a ∈ A. card (above (limit A r) a) ≤ 1}
        by auto
      thus ?thesis
        by simp
    qed
    hence defer (pass-module 1 r) A p = {w}
      by simp
    thus card (defer (pass-module 1 r) A p) = 1
      by simp
   qed
  qed
 qed
 thus ?thesis
  using defers-def order pass-mod-sound
  by blast
qed

theorem pass-two-mod-def-two:
 assumes order: linear-order r
 shows defers 2 (pass-module 2 r)
 unfolding defers-def
proof (safe)
 show electoral-module (pass-module 2 r)
  using order
  by simp
next
 fix
  A :: 'a set and
  p :: 'a Profile
 assume
  min-2-card: 2 ≤ card A and
  finA: finite A and
  profA: profile A p
 from min-2-card
```

have *not-empty-A*: $A \neq \{\}$
by *auto*
moreover have *limitA-order*:
linear-order-on A (*limit* A r)
using *limit-presv-lin-ord order*
by *auto*
ultimately obtain a **where**
a: *above* (*limit* A r) $a = \{a\}$
using *above-one min-2-card finA profA*
by *blast*
hence $\forall\, b \in A$. *let* q = *limit* A r *in* $(b \preceq_q a)$
using *limitA-order pref-imp-in-above empty-iff*
insert-iff insert-subset above-presv-limit
order connex-def lin-ord-imp-connex
by *metis*
hence *a-best*: $\forall\, b \in A$. $(b, a) \in$ *limit* A r
by *simp*
hence *a-above*: $\forall\, b \in A$. $a \in$ *above* (*limit* A r) b
by (*simp add*: *above-def*)
from a **have** $a \in \{a \in A$. *card*(*above* (*limit* A r) $a) \leq 2\}$
using *CollectI Suc-leI not-empty-A a-above card-UNIV-bool*
card-eq-0-iff card-insert-disjoint empty-iff finA
finite.emptyI insert-iff limitA-order above-one
UNIV-bool nat.simps(*3*) *zero-less-Suc*
by (*metis* (*no-types*, *lifting*))
hence *a-in-defer*: $a \in$ *defer* (*pass-module 2* r) A p
by *simp*
have *finite* $(A - \{a\})$
by (*simp add*: *finA*)
moreover have *A-not-only-a*: $A - \{a\} \neq \{\}$
using *min-2-card Diff-empty Diff-idemp Diff-insert0*
One-nat-def not-empty-A card.insert-remove
card-eq-0-iff finite.emptyI insert-Diff
numeral-le-one-iff semiring-norm(*69*) *card.empty*
by *metis*
moreover have *limitAa-order*:
linear-order-on $(A - \{a\})$ (*limit* $(A - \{a\})$ r)
using *limit-presv-lin-ord order top-greatest*
by *blast*
ultimately obtain b **where** b: *above* (*limit* $(A - \{a\})$ r) $b = \{b\}$
using *above-one*
by *metis*
hence $\forall\, c \in A - \{a\}$. *let* q = *limit* $(A - \{a\})$ r *in* $(c \preceq_q b)$
using *limitAa-order pref-imp-in-above empty-iff insert-iff*
insert-subset above-presv-limit order connex-def
lin-ord-imp-connex
by *metis*
hence *b-in-limit*: $\forall\, c \in A - \{a\}$. $(c, b) \in$ *limit* $(A - \{a\})$ r
by *simp*

hence *b-best*: $\forall\, c \in A-\{a\}.\ (c,\ b) \in$ *limit A r*
  by *auto*
hence *c-not-above-b*: $\forall\, c \in A-\{a,\ b\}.\ c \notin$ *above* (*limit A r*) *b*
  using *b Diff-iff Diff-insert2 subset-UNIV above-presv-limit*
    insert-subset order limit-presv-above limit-presv-above2
  by *metis*
moreover have *above-subset*: *above* (*limit A r*) *b* $\subseteq A$
  using *above-presv-limit order*
  by *metis*
moreover have *b-above-b*: $b \in$ *above* (*limit A r*) *b*
  using *above-def b b-best above-presv-limit*
    mem-Collect-eq order insert-subset
  by *metis*
ultimately have *above-b-eq-ab*: *above* (*limit A r*) $b = \{a,\ b\}$
  using *a-above*
  by *auto*
hence *card-above-b-eq-2*: *card* (*above* (*limit A r*) *b*) = *2*
  using *A-not-only-a b-in-limit*
  by *auto*
hence *b-in-defer*: $b \in$ *defer* (*pass-module 2 r*) *A p*
  using *b-above-b above-subset*
  by *auto*
from *b-best* **have** *b-above*:
  $\forall\, c \in A-\{a\}.\ b \in$ *above* (*limit A r*) *c*
  using *above-def mem-Collect-eq*
  by *metis*
have *connex A* (*limit A r*)
  using *limitA-order lin-ord-imp-connex*
  by *auto*
hence $\forall\, c \in A.\ c \in$ *above* (*limit A r*) *c*
  by (*simp add*: *above-connex*)
hence $\forall\, c \in A-\{a,\ b\}.\ \{a,\ b,\ c\} \subseteq$ *above* (*limit A r*) *c*
  using *a-above b-above*
  by *auto*
moreover have $\forall\, c \in A-\{a,\ b\}.$ *card*$\{a,\ b,\ c\}$ = *3*
  using *DiffE One-nat-def Suc-1 above-b-eq-ab card-above-b-eq-2*
    above-subset card-insert-disjoint finA finite-subset
    insert-commute numeral-3-eq-3
  by *metis*
ultimately have
  $\forall\, c \in A-\{a,\ b\}.$ *card* (*above* (*limit A r*) *c*) $\geq$ *3*
  using *card-mono finA finite-subset above-presv-limit order*
  by *metis*
hence $\forall\, c \in A-\{a,\ b\}.$ *card* (*above* (*limit A r*) *c*) > *2*
  using *less-le-trans numeral-less-iff order-refl semiring-norm*(*79*)
  by *metis*
hence $\forall\, c \in A-\{a,\ b\}.\ c \notin$ *defer* (*pass-module 2 r*) *A p*
  by (*simp add*: *not-le*)
moreover have *defer* (*pass-module 2 r*) *A p* $\subseteq A$

by *auto*
ultimately have *defer* (*pass-module 2 r*) *A p* $\subseteq$ {*a*, *b*}
by *blast*
with *a-in-defer b-in-defer* **have**
defer (*pass-module 2 r*) *A p* = {*a*, *b*}
by *fastforce*
thus *card* (*defer* (*pass-module 2 r*) *A p*) = *2*
using *above-b-eq-ab card-above-b-eq-2*
by *presburger*
qed

end

3.4 Elect Module

theory *Elect-Module*
imports *Component-Types/Electoral-Module*
begin

The elect module is not concerned about the voter's ballots, and just elects all alternatives. It is primarily used in sequence after an electoral module that only defers alternatives to finalize the decision, thereby inducing a proper voting rule in the social choice sense.

3.4.1 Definition

fun *elect-module* :: *'a Electoral-Module* **where**
elect-module A p = (*A*, {}, {})

3.4.2 Soundness

theorem *elect-mod-sound*[*simp*]: *electoral-module elect-module*
unfolding *electoral-module-def*
by *simp*

3.4.3 Electing

theorem *elect-mod-electing*[*simp*]: *electing elect-module*
unfolding *electing-def*
by *simp*

end

3.5 Plurality Module

theory *Plurality-Module*
  imports *Component-Types/Electoral-Module*
begin

The plurality module implements the plurality voting rule. The plurality rule elects all modules with the maximum amount of top preferences among all alternatives, and rejects all the other alternatives. It is electing and induces the classical plurality (voting) rule from social-choice theory.

3.5.1 Definition

fun *plurality* :: *'a Electoral-Module* **where**
  plurality A p =
    $(\{a \in A.\ \forall\, x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\},$
     $\{a \in A.\ \exists\, x \in A.\ \textit{win-count}\ p\ x > \textit{win-count}\ p\ a\},$
     $\{\})$

fun *plurality-code* :: *'a Electoral-Module* **where**
  plurality-code A p =
    $(\{a \in A.\ \forall\, x \in A.\ \textit{win-count-code}\ p\ x \leq \textit{win-count-code}\ p\ a\},$
     $\{a \in A.\ \exists\, x \in A.\ \textit{win-count-code}\ p\ x > \textit{win-count-code}\ p\ a\},$
     $\{\})$

3.5.2 Soundness

theorem *plurality-sound*[*simp*]: *electoral-module plurality*
proof −
  have
    $\forall\, A\ p.$
      $\textit{let elect} = \{a \in (A::'a\ \textit{set}).\ \forall\, x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\};$
      $\textit{reject} = \{a \in A.\ \exists\, x \in A.\ \textit{win-count}\ p\ x > \textit{win-count}\ p\ a\}\ \textit{in}$
    $\textit{elect} \cap \textit{reject} = \{\}$
    by *auto*
  hence *disjoint*:
    $\forall\, A\ p.$
      $\textit{let elect} = \{a \in (A::'a\ \textit{set}).\ \forall\, x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\};$
      $\textit{reject} = \{a \in A.\ \exists\, x \in A.\ \textit{win-count}\ p\ x > \textit{win-count}\ p\ a\}\ \textit{in}$
    $\textit{disjoint3}\ (\textit{elect}, \textit{reject}, \{\})$
    by *simp*
  have
    $\forall\, A\ p.$
      $\textit{let elect} = \{a \in (A::'a\ \textit{set}).\ \forall\, x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\};$
      $\textit{reject} = \{a \in A.\ \exists\, x \in A.\ \textit{win-count}\ p\ x > \textit{win-count}\ p\ a\}\ \textit{in}$
    $\textit{elect} \cup \textit{reject} = A$

using *not-le-imp-less*
by *auto*
hence *unity*:
$\forall A\ p.$
let elect = $\{a \in (A::'a\ set).\ \forall x \in A.\ win\text{-}count\ p\ x \leq win\text{-}count\ p\ a\}$;
reject = $\{a \in A.\ \exists x \in A.\ win\text{-}count\ p\ x > win\text{-}count\ p\ a\}$ *in*
set-equals-partition A (*elect*, *reject*, $\{\}$)
by *simp*
from *disjoint unity* **show** *?thesis*
by (*simp add*: *electoral-modI*)
qed

3.5.3 Electing

lemma *plurality-electing2*: $\forall A\ p.$
$(A \neq \{\} \wedge finite\text{-}profile\ A\ p) \longrightarrow$
$elect\ plurality\ A\ p \neq \{\}$
proof (*intro allI impI conjI*)
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$
assume
assm0: $A \neq \{\} \wedge finite\text{-}profile\ A\ p$
show
$elect\ plurality\ A\ p \neq \{\}$
proof
obtain *max* **where**
max: $max = Max(win\text{-}count\ p\ `\ A)$
by *simp*
then obtain a **where**
a: $win\text{-}count\ p\ a = max \wedge a \in A$
using *Max-in assm0 empty-is-image*
finite-imageI imageE
by (*metis* (*no-types*, *lifting*))
hence
$\forall x \in A.\ win\text{-}count\ p\ x \leq win\text{-}count\ p\ a$
by (*simp add*: *max assm0*)
moreover have
$a \in A$
using a
by *simp*
ultimately have
$a \in \{a \in A.\ \forall x \in A.\ win\text{-}count\ p\ x \leq win\text{-}count\ p\ a\}$
by *blast*
hence *a-elem*:
$a \in elect\ plurality\ A\ p$
by *simp*
assume
assm1: $elect\ plurality\ A\ p = \{\}$

```
    thus False
      using a-elem assm1 all-not-in-conv
      by metis
  qed
qed


theorem plurality-electing[simp]: electing plurality
proof −
  have electoral-module plurality ∧
      (∀ A p. (A ≠ {} ∧ finite-profile A p) ⟶ elect plurality A p ≠ {})
  proof
    show electoral-module plurality
      by simp
  next
    show (∀ A p. (A ≠ {} ∧ finite-profile A p) ⟶ elect plurality A p ≠ {})
      using plurality-electing2
      by metis
  qed
  thus ?thesis
      by (simp add: electing-def)
qed
```

3.5.4 Property

```
lemma plurality-inv-mono2: ∀ A p q a.
                              (a ∈ elect plurality A p ∧ lifted A p q a) ⟶
                                (elect plurality A q = elect plurality A p ∨
                                    elect plurality A q = {a})
proof (intro allI impI)
  fix
    A :: 'a set and
    p :: 'a Profile and
    q :: 'a Profile and
    a :: 'a
  assume
    asm1:
    a ∈ elect plurality A p ∧ lifted A p q a
  show
    elect plurality A q = elect plurality A p ∨
        elect plurality A q = {a}
  proof −
    have lifted-winner:
      ∀ x ∈ A.
         ∀ i::nat. i < size p ⟶
           (above (p!i) x = {x} ⟶
              (above (q!i) x = {x} ∨ above (q!i) a = {a}))
      using asm1 Profile.lifted-def lifted-above-winner
      by (metis (no-types, lifting))
```

hence
  $\forall\, i::nat.\ i < size\ p \longrightarrow$
    $(above\ (p!i)\ a = \{a\} \longrightarrow above\ (q!i)\ a = \{a\})$
  using *asm1*
  by *auto*
hence *a-win-subset*:
  $\{i::nat.\ i < size\ p \wedge above\ (p!i)\ a = \{a\}\} \subseteq$
    $\{i::nat.\ i < size\ p \wedge above\ (q!i)\ a = \{a\}\}$
  by *blast*
moreover have *sizes*:
  $size\ p = size\ q$
  using *asm1 Profile.lifted-def*
  by *metis*
ultimately have *win-count-a*:
  $win\text{-}count\ p\ a \leq win\text{-}count\ q\ a$
  by (*simp add*: *card-mono*)
have *fin-A*:
  finite A
  using *asm1 Profile.lifted-def*
  by *metis*
hence
  $\forall\, x \in A-\{a\}.$
    $\forall\, i::nat.\ i < size\ p \longrightarrow$
    $(above\ (q!i)\ a = \{a\} \longrightarrow above\ (q!i)\ x \neq \{x\})$
  using *DiffE Profile.lifted-def above-one2*
      asm1 insertCI insert-absorb insert-not-empty
      profile-def sizes
  by *metis*
with *lifted-winner* **have** *above-QtoP*:
  $\forall\, x \in A-\{a\}.$
    $\forall\, i::nat.\ i < size\ p \longrightarrow$
    $(above\ (q!i)\ x = \{x\} \longrightarrow above\ (p!i)\ x = \{x\})$
  using *lifted-above-winner3 asm1*
      Profile.lifted-def
  by *metis*
hence
  $\forall\, x \in A-\{a\}.$
    $\{i::nat.\ i < size\ p \wedge above\ (q!i)\ x = \{x\}\} \subseteq$
    $\{i::nat.\ i < size\ p \wedge above\ (p!i)\ x = \{x\}\}$
  by (*simp add*: *Collect-mono*)
hence *win-count-other*:
  $\forall\, x \in A-\{a\}.\ win\text{-}count\ p\ x \geq win\text{-}count\ q\ x$
  by (*simp add*: *card-mono sizes*)
show
  $elect\ plurality\ A\ q = elect\ plurality\ A\ p\ \vee$
    $elect\ plurality\ A\ q = \{a\}$
proof *cases*
  assume $win\text{-}count\ p\ a = win\text{-}count\ q\ a$
  hence

$card\ \{i::nat.\ i < size\ p \wedge above\ (p!i)\ a = \{a\}\} =$
$\quad card\ \{i::nat.\ i < size\ p \wedge above\ (q!i)\ a = \{a\}\}$
by ($simp\ add$: $sizes$)
moreover have
$finite\ \{i::nat.\ i < size\ p \wedge above\ (q!i)\ a = \{a\}\}$
by $simp$
ultimately have
$\{i::nat.\ i < size\ p \wedge above\ (p!i)\ a = \{a\}\} =$
$\quad \{i::nat.\ i < size\ p \wedge above\ (q!i)\ a = \{a\}\}$
using $a\text{-}win\text{-}subset$
by ($simp\ add$: $card\text{-}subset\text{-}eq$)
hence $above\text{-}pq$:
$\forall i::nat.\ i < size\ p \longrightarrow$
$\quad above\ (p!i)\ a = \{a\} \longleftrightarrow above\ (q!i)\ a = \{a\}$
by $blast$
moreover have
$\forall x \in A-\{a\}.$
$\quad \forall i::nat.\ i < size\ p \longrightarrow$
$\quad\quad (above\ (p!i)\ x = \{x\} \longrightarrow$
$\quad\quad\quad (above\ (q!i)\ x = \{x\} \vee above\ (q!i)\ a = \{a\}))$
using $lifted\text{-}winner$
by $auto$
moreover have
$\forall x \in A-\{a\}.$
$\quad \forall i::nat.\ i < size\ p \longrightarrow$
$\quad\quad (above\ (p!i)\ x = \{x\} \longrightarrow above\ (p!i)\ a \neq \{a\})$
proof ($rule\ ccontr$)
assume
$\neg(\forall x \in A-\{a\}.$
$\quad \forall i::nat.\ i < size\ p \longrightarrow$
$\quad\quad (above\ (p!i)\ x = \{x\} \longrightarrow above\ (p!i)\ a \neq \{a\}))$
hence
$\exists x \in A-\{a\}.$
$\quad \exists i::nat.$
$\quad\quad i < size\ p \wedge above\ (p!i)\ x = \{x\} \wedge above\ (p!i)\ a = \{a\}$
by $auto$
moreover from $this$ **have**
$finite\ A \wedge A \neq \{\}$
using $fin\text{-}A$
by $blast$
moreover from $asm1$ **have**
$\forall i::nat.\ i < size\ p \longrightarrow linear\text{-}order\text{-}on\ A\ (p!i)$
by ($simp\ add$: $Profile.lifted\text{-}def\ profile\text{-}def$)
ultimately have
$\exists x \in A-\{a\}.\ x = a$
using $above\text{-}one2$
by $metis$
thus $False$
by $simp$

qed
ultimately have *above-PtoQ*:
$\forall x \in A-\{a\}.$
$\forall i{::}nat.\ i < size\ p \longrightarrow$
$(above\ (p!i)\ x = \{x\} \longrightarrow above\ (q!i)\ x = \{x\})$
by (*simp add*: *above-pq*)
hence
$\forall x \in A.$
$card\ \{i{::}nat.\ i < size\ p \land above\ (p!i)\ x = \{x\}\} =$
$card\ \{i{::}nat.\ i < size\ q \land above\ (q!i)\ x = \{x\}\}$
using *Collect-cong DiffI above-pq above-QtoP*
insert-absorb insert-iff insert-not-empty sizes
by (*smt* (*verit*, *ccfv-threshold*))
hence $\forall x \in A.\ win\text{-}count\ p\ x = win\text{-}count\ q\ x$
by *simp*
hence
$\{a \in A.\ \forall x \in A.\ win\text{-}count\ p\ x \leq win\text{-}count\ p\ a\} =$
$\{a \in A.\ \forall x \in A.\ win\text{-}count\ q\ x \leq win\text{-}count\ q\ a\}$
by *auto*
thus *?thesis*
by *simp*
next
assume $win\text{-}count\ p\ a \neq win\text{-}count\ q\ a$
hence *strict-less*:
$win\text{-}count\ p\ a < win\text{-}count\ q\ a$
using *win-count-a*
by *auto*
have *a-in-win-p*:
$a \in \{a \in A.\ \forall x \in A.\ win\text{-}count\ p\ x \leq win\text{-}count\ p\ a\}$
using *asm1*
by *auto*
hence $\forall x \in A.\ win\text{-}count\ p\ x \leq win\text{-}count\ p\ a$
by *simp*
with *strict-less win-count-other*
have *less*:
$\forall x \in A-\{a\}.\ win\text{-}count\ q\ x < win\text{-}count\ q\ a$
using *DiffD1 antisym dual-order.trans*
not-le-imp-less win-count-a
by *metis*
hence
$\forall x \in A-\{a\}.\ \neg(\forall y \in A.\ win\text{-}count\ q\ y \leq win\text{-}count\ q\ x)$
using *asm1 Profile.lifted-def not-le*
by *metis*
hence
$\forall x \in A-\{a\}.$
$x \notin \{a \in A.\ \forall x \in A.\ win\text{-}count\ q\ x \leq win\text{-}count\ q\ a\}$
by *blast*
hence
$\forall x \in A-\{a\}.\ x \notin elect\ plurality\ A\ q$

by *simp*
moreover have
$a \in$ *elect plurality* $A\ q$
proof −
from *less*
have
$\forall x \in A - \{a\}$. *win-count* $q\ x \leq$ *win-count* $q\ a$
using *less-imp-le*
by *metis*
moreover have
win-count $q\ a \leq$ *win-count* $q\ a$
by *simp*
ultimately have
$\forall x \in A$. *win-count* $q\ x \leq$ *win-count* $q\ a$
by *auto*
moreover have
$a \in A$
using *a-in-win-p*
by *auto*
ultimately have
$a \in \{a \in A.$
$\forall x \in A$. *win-count* $q\ x \leq$ *win-count* $q\ a\}$
by *auto*
thus *?thesis*
by *simp*
qed
moreover have
elect plurality $A\ q \subseteq A$
by *simp*
ultimately show *?thesis*
by *auto*
qed
qed
qed

theorem *plurality-inv-mono*[*simp*]: *invariant-monotonicity plurality*
proof −
have
electoral-module plurality $\wedge$
$(\forall A\ p\ q\ a.$
$(a \in$ *elect plurality* $A\ p \wedge$ *lifted* $A\ p\ q\ a) \longrightarrow$
$($*elect plurality* $A\ q =$ *elect plurality* $A\ p \vee$
elect plurality $A\ q = \{a\}))$
proof
show *electoral-module plurality*
by *simp*
next
show

$\forall A\ p\ q\ a.\ (a \in elect\ plurality\ A\ p \wedge lifted\ A\ p\ q\ a) \longrightarrow$
$(elect\ plurality\ A\ q = elect\ plurality\ A\ p \vee$
$elect\ plurality\ A\ q = \{a\})$
using *plurality-inv-mono2*
by *metis*
qed
thus *?thesis*
by (*simp add*: *invariant-monotonicity-def*)
qed

end

3.6 Borda Module

theory *Borda-Module*
imports *Component-Types/Elimination-Module*
begin

This is the Borda module used by the Borda rule. The Borda rule is a voting rule, where on each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

3.6.1 Definition

fun *borda-score* :: *'a Evaluation-Function* **where**
borda-score $x\ A\ p = (\sum y \in A.\ (prefer\text{-}count\ p\ x\ y))$

fun *borda-score-code* :: *'a Evaluation-Function* **where**
borda-score-code $x\ A\ p = (\sum y \in A.\ (prefer\text{-}count\text{-}code\ p\ x\ y))$

fun *borda* :: *'a Electoral-Module* **where**
borda $A\ p$ = *max-eliminator borda-score* $A\ p$

fun *borda-code* :: *'a Electoral-Module* **where**
borda-code $A\ p$ = (*max-eliminator borda-score-code*) $A\ p$

end

3.7 Condorcet Module

theory *Condorcet-Module*
  imports *Component-Types/Elimination-Module*
begin

This is the Condorcet module used by the Condorcet (voting) rule. The Condorcet rule is a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

3.7.1 Definition

```
fun condorcet-score :: 'a Evaluation-Function where
  condorcet-score x A p =
    (if (condorcet-winner A p x) then 1 else 0)

fun condorcet-score-code :: 'a Evaluation-Function where
  condorcet-score-code x A p =
    (if (condorcet-winner-code A p x) then 1 else 0)

fun condorcet :: 'a Electoral-Module where
  condorcet A p = (max-eliminator condorcet-score) A p

fun condorcet-code :: 'a Electoral-Module where
  condorcet-code A p =
    (max-eliminator condorcet-score-code) A p
```

3.7.2 Property

```
theorem condorcet-score-is-condorcet-rating: condorcet-rating condorcet-score
proof –
  have
    ∀ f.
      (¬ condorcet-rating f ⟶
         (∃ A rs a.
           condorcet-winner A rs a ∧
             (∃ aa. ¬ f (aa::'a) A rs < f a A rs ∧ a ≠ aa ∧ aa ∈ A))) ∧
        (condorcet-rating f ⟶
           (∀ A rs a. condorcet-winner A rs a ⟶
             (∀ aa. f aa A rs < f a A rs ∨ a = aa ∨ aa ∉ A)))
    unfolding condorcet-rating-def
    by (metis (mono-tags, hide-lams))
  thus ?thesis
    using cond-winner-unique condorcet-score.simps zero-less-one
    by (metis (no-types))
qed
```

theorem *condorcet-is-dcc*: *defer-condorcet-consistency condorcet*
proof −
  have *max-cscore-dcc*:
    defer-condorcet-consistency (*max-eliminator condorcet-score*)
    using *cr-eval-imp-dcc-max-elim*
    by (*simp add*: *condorcet-score-is-condorcet-rating*)
  have *cond-eq-max-cond*:
    $\bigwedge A\ p.$ (*condorcet* A $p \equiv$ *max-eliminator condorcet-score* A p)
    by *simp*
  from *max-cscore-dcc cond-eq-max-cond* **show** *?thesis*
    unfolding *defer-condorcet-consistency-def electoral-module-def*
    by (*smt* (*verit*, *ccfv-threshold*))
qed

end

3.8 Copeland Module

theory *Copeland-Module*
  imports *Component-Types/Elimination-Module*
begin

This is the Copeland module used by the Copeland voting rule. The Copeland rule elects the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

3.8.1 Definition

fun *copeland-score* :: *'a Evaluation-Function* **where**
  copeland-score x A p =
    card$\{y \in A$. *wins* x p $y\}$ − *card*$\{y \in A$. *wins* y p $x\}$

fun *copeland-score-code* :: *'a Evaluation-Function* **where**
  copeland-score-code x A p =
    card$\{y \in A$. *wins-code* x p $y\}$ − *card*$\{y \in A$. *wins-code* y p $x\}$

fun *copeland* :: *'a Electoral-Module* **where**
  copeland A p = *max-eliminator copeland-score* A p

fun *copeland-code* :: *'a Electoral-Module* **where**
  copeland-code A p =

max-eliminator copeland-score-code A p

3.8.2 Lemmata

lemma *cond-winner-imp-win-count*:
assumes *winner*: *condorcet-winner A p w*
shows *card* $\{y \in A$. *wins w p y*$\}$ = *card A* −*1*
proof −
from *winner*
have *0*: $\forall x \in A - \{w\}$. *wins w p x*
by *simp*
have *1*: $\forall M$. $\{x \in M$. *True*$\} = M$
by *blast*
from *0 1*
have $\{x \in A - \{w\}$. *wins w p x*$\} = A - \{w\}$
by *blast*
hence *10*: *card* $\{x \in A - \{w\}$. *wins w p x*$\}$ = *card* $(A - \{w\})$
by *simp*
from *winner*
have *11*: $w \in A$
by *simp*
hence *card* $(A - \{w\})$ = *card A* − *1*
using *card-Diff-singleton condorcet-winner.simps winner*
by *metis*
hence *amount1*:
card $\{x \in A - \{w\}$. *wins w p x*$\}$ = *card* (A) − *1*
using *10*
by *linarith*
have *2*: $\forall x \in \{w\}$. ¬ *wins x p x*
by (*simp add*: *wins-irreflex*)
have *3*: $\forall M$. $\{x \in M$. *False*$\} = \{\}$
by *blast*
from *2 3*
have $\{x \in \{w\}$. *wins w p x*$\} = \{\}$
by *blast*
hence *amount2*: *card* $\{x \in \{w\}$. *wins w p x*$\}$ = *0*
by *simp*
have *disjunct*:
$\{x \in A - \{w\}$. *wins w p x*$\} \cap \{x \in \{w\}$. *wins w p x*$\} = \{\}$
by *blast*
have *union*:
$\{x \in A - \{w\}$. *wins w p x*$\} \cup \{x \in \{w\}$. *wins w p x*$\}$ =
$\{x \in A$. *wins w p x*$\}$
using *2*
by *blast*
have *finiteness1*: *finite* $\{x \in A - \{w\}$. *wins w p x*$\}$
using *condorcet-winner.simps winner*
by *fastforce*
have *finiteness2*: *finite* $\{x \in \{w\}$. *wins w p x*$\}$

by *simp*
from *finiteness1 finiteness2 disjunct card-Un-disjoint*
have
$card\ (\{x \in A - \{w\}\ .\ wins\ w\ p\ x\} \cup \{x \in \{w\}\ .\ wins\ w\ p\ x\}) =$
$card\ \{x \in A - \{w\}\ .\ wins\ w\ p\ x\} + card\ \{x \in \{w\}\ .\ wins\ w\ p\ x\}$
by *blast*
with *union*
have $card\ \{x \in A\ .\ wins\ w\ p\ x\} =$
$card\ \{x \in A - \{w\}\ .\ wins\ w\ p\ x\} + card\ \{x \in \{w\}\ .\ wins\ w\ p\ x\}$
by *simp*
with *amount1 amount2*
show *?thesis*
by *linarith*
qed

lemma *cond-winner-imp-loss-count*:
assumes *winner*: *condorcet-winner A p w*
shows $card\ \{y \in A\ .\ wins\ y\ p\ w\} = 0$
using *Collect-empty-eq card-eq-0-iff condorcet-winner.simps*
insert-Diff insert-iff wins-antisym winner
by (*metis* (*no-types*, *lifting*))

lemma *cond-winner-imp-copeland-score*:
assumes *winner*: *condorcet-winner A p w*
shows $copeland\text{-}score\ w\ A\ p = card\ A\ - 1$
unfolding *copeland-score.simps*
proof −
show
$card\ \{y \in A.\ wins\ w\ p\ y\} - card\ \{y \in A.\ wins\ y\ p\ w\} =$
$card\ A - 1$
using *cond-winner-imp-loss-count*
cond-winner-imp-win-count winner
proof −
have *f1*: $card\ \{a \in A.\ wins\ w\ p\ a\} = card\ A - 1$
using *cond-winner-imp-win-count winner*
by *simp*
have *f2*: $card\ \{a \in A.\ wins\ a\ p\ w\} = 0$
using *cond-winner-imp-loss-count winner*
by (*metis* (*no-types*))
have $card\ A - 1 - 0 = card\ A - 1$
by *simp*
thus *?thesis*
using *f2 f1*
by *simp*
qed
qed

```
lemma non-cond-winner-imp-win-count:
  assumes
    winner: condorcet-winner A p w and
    loser: l ≠ w and
    l-in-A: l ∈ A
  shows card {y ∈ A . wins l p y} <= card A − 2
proof −
  from winner loser l-in-A
  have wins w p l
    by simp
  hence 0: ¬ wins l p w
    by (simp add: wins-antisym)
  have 1: ¬ wins l p l
    by (simp add: wins-irreflex)
  from 0 1 have 2:
    {y ∈ A . wins l p y} =
        {y ∈ A−{l,w} . wins l p y}
    by blast
  have 3: ∀ M f . finite M ⟶ card {x ∈ M . f x} ≤ card M
    by (simp add: card-mono)
  have 4: finite (A−{l,w})
    using condorcet-winner.simps finite-Diff winner
    by metis
  from 3 4 have 5:
    card {y ∈ A−{l,w} . wins l p y} ≤
      card (A−{l,w})
    by (metis (full-types))
  have w ∈ A
    using condorcet-winner.simps winner
    by metis
  with l-in-A
  have card(A−{l,w}) = card A − card {l,w}
    by (simp add: card-Diff-subset)
  hence card(A−{l,w}) = card A − 2
    by (simp add: loser)
  with 5 2
  show ?thesis
    by simp
qed
```

3.8.3 Property

```
theorem copeland-score-is-cr: condorcet-rating copeland-score
  unfolding condorcet-rating-def
proof (unfold copeland-score.simps, safe)
  fix
    A :: 'a set and
    p :: 'a Profile and
```

```
    w :: 'a and
    l :: 'a
  assume
    winner: condorcet-winner A p w and
    l-in-A: l ∈ A and
    l-neq-w: l ≠ w
  show
    card {y ∈ A. wins l p y} − card {y ∈ A. wins y p l}
        < card {y ∈ A. wins w p y} − card {y ∈ A. wins y p w}
  proof −
    from winner have 0:
      card {y ∈ A. wins w p y} − card {y ∈ A. wins y p w} =
        card A −1
      using cond-winner-imp-copeland-score
      by fastforce
    from winner l-neq-w l-in-A have 1:
      card {y ∈ A. wins l p y} − card {y ∈ A. wins y p l} ≤
          card A −2
      using non-cond-winner-imp-win-count
      by fastforce
    have 2: card A −2 < card A −1
      using card-0-eq card-Diff-singleton
            condorcet-winner.simps diff-less-mono2
            empty-iff finite-Diff insertE insert-Diff
            l-in-A l-neq-w neq0-conv one-less-numeral-iff
            semiring-norm(76) winner zero-less-diff
      by metis
    hence
      card {y ∈ A. wins l p y} − card {y ∈ A. wins y p l} <
        card A −1
      using 1 le-less-trans
      by blast
    with 0
    show ?thesis
      by linarith
  qed
qed

theorem copeland-is-dcc: defer-condorcet-consistency copeland
proof −
  have max-cplscore-dcc:
    defer-condorcet-consistency (max-eliminator copeland-score)
    using cr-eval-imp-dcc-max-elim
    by (simp add: copeland-score-is-cr)
  have copel-eq-max-copel:
    ⋀A p. (copeland A p ≡ max-eliminator copeland-score A p)
    by simp
  from max-cplscore-dcc copel-eq-max-copel
  show ?thesis
```

unfolding *defer-condorcet-consistency-def electoral-module-def*
by (*smt* (*verit*, *ccfv-threshold*))
qed

end

3.9 Minimax Module

theory *Minimax-Module*
imports *Component-Types/Elimination-Module*
begin

This is the Minimax module used by the Minimax voting rule. The Minimax rule elects the alternatives with the highest Minimax score. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

3.9.1 Definition

fun *minimax-score* :: *'a Evaluation-Function* **where**
minimax-score x A p =
Min {*prefer-count p x y* |*y* . *y* ∈ *A*−{*x*}}

fun *minimax-score-code* :: *'a Evaluation-Function* **where**
minimax-score-code x A p =
Min {*prefer-count-code p x y* |*y* . *y* ∈ *A*−{*x*}}

fun *minimax* :: *'a Electoral-Module* **where**
minimax A p = *max-eliminator minimax-score A p*

fun *minimax-code* :: *'a Electoral-Module* **where**
minimax-code A p = *max-eliminator minimax-score-code A p*

3.9.2 Lemma

lemma *non-cond-winner-minimax-score*:
assumes
prof: *profile A p* **and**
winner: *condorcet-winner A p w* **and**
l-in-A: *l* ∈ *A* **and**
l-neq-w: *l* ≠ *w*
shows *minimax-score l A p* ≤ *prefer-count p l w*
proof −
let

?set = {*prefer-count p l y* |*y* . *y* ∈ *A*−{*l*}} **and**
?lscore = *minimax-score l A p*
have *finite A*
using *prof condorcet-winner.simps winner*
by *metis*
hence *finite* (*A*−{*l*})
using *finite-Diff*
by *simp*
hence *finite*: *finite ?set*
by *simp*
have *w* ∈ *A*
using *condorcet-winner.simps winner*
by *metis*
hence *0*: *w* ∈ *A*−{*l*}
using *l-neq-w*
by *force*
hence *not-empty*: *?set* ≠ {}
by *blast*

have *?lscore* = *Min ?set*
by *simp*
hence *1*: *?lscore* ∈ *?set* ∧ (∀ *p* ∈ *?set*. *?lscore* ≤ *p*)
using *local.finite not-empty Min-le Min-eq-iff*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
using *0*
by *auto*
qed

3.9.3 Property

theorem *minimax-score-cond-rating*: *condorcet-rating minimax-score*
proof (*unfold condorcet-rating-def minimax-score.simps prefer-count.simps*, *safe*)
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a* **and**
l :: *'a*
assume
winner: *condorcet-winner A p w* **and**
l-in-A: *l* ∈ *A* **and**
l-neq-w:*l* ≠ *w*
show
$Min\ \{card\ \{i.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r l))\}\ |$
$y.\ y \in A - \{l\}\} <$
$Min\ \{card\ \{i.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r w))\}\ |$
$y.\ y \in A - \{w\}\}$
proof (*rule ccontr*)
assume

$\neg$ *Min* $\{$*card* $\{i.\ i <$ *length* $p \wedge ($*let* $r = (p!i)$ *in* $(y \preceq_r l))\}$ |
  $y.\ y \in A - \{l\}\} <$
 Min $\{$*card* $\{i.\ i <$ *length* $p \wedge ($*let* $r = (p!i)$ *in* $(y \preceq_r w))\}$ |
  $y.\ y \in A - \{w\}\}$
hence
 Min $\{$*card* $\{i.\ i <$ *length* $p \wedge ($*let* $r = (p!i)$ *in* $(y \preceq_r l))\}$ |
  $y.\ y \in A - \{l\}\} \geq$
 Min $\{$*card* $\{i.\ i <$ *length* $p \wedge ($*let* $r = (p!i)$ *in* $(y \preceq_r w))\}$ |
  $y.\ y \in A - \{w\}\}$
 by *linarith*
hence *000*:
 Min $\{$*prefer-count* $p\ l\ y\ |y\ .\ y \in A-\{l\}\} \geq$
 Min $\{$*prefer-count* $p\ w\ y\ |y\ .\ y \in A-\{w\}\}$
 by *auto*
have *prof*: *profile* $A\ p$
 using *condorcet-winner.simps winner*
 by *metis*
from *prof winner l-in-A l-neq-w*
have *100*:
 prefer-count $p\ l\ w \geq$ *Min* $\{$*prefer-count* $p\ l\ y\ |y\ .\ y \in A-\{l\}\}$
 using *non-cond-winner-minimax-score minimax-score.simps*
 by *metis*

from *l-in-A*
have *l-in-A-without-w*: $l \in A-\{w\}$
 by (*simp add*: *l-neq-w*)
hence *2*: $\{$*prefer-count* $p\ w\ y\ |y\ .\ y \in A-\{w\}\} \neq \{\}$
 by *blast*
have *finite* $(A-\{w\})$
 using *prof condorcet-winner.simps winner finite-Diff*
 by *metis*
hence *3*: *finite* $\{$*prefer-count* $p\ w\ y\ |y\ .\ y \in A-\{w\}\}$
 by *simp*
from *2 3*
have *4*:
 $\exists\ n \in A-\{w\}$. *prefer-count* $p\ w\ n =$
 Min $\{$*prefer-count* $p\ w\ y\ |y\ .\ y \in A-\{w\}\}$
 using *Min-in*
 by *fastforce*
then obtain n **where** *200*:
 prefer-count $p\ w\ n =$
 Min $\{$*prefer-count* $p\ w\ y\ |y\ .\ y \in A-\{w\}\}$ **and**
 6: $n \in A-\{w\}$
 by *metis*
hence *n-in-A*: $n \in A$
 using *DiffE*
 by *metis*
from *6*
have *n-neq-w*: $n \neq w$

```
      by auto
    from winner
    have w-in-A: w ∈ A
      by simp
    from 6 prof winner
    have 300: prefer-count p w n > prefer-count p n w
      by simp
    from 100 000 200
    have 400: prefer-count p l w ≥ prefer-count p w n
      by linarith
    with prof n-in-A w-in-A l-in-A n-neq-w
        l-neq-w pref-count-sym
    have 700: prefer-count p n w ≥ prefer-count p w l
      by metis
    have prefer-count p l w > prefer-count p w l
      using 300 400 700
      by linarith
    hence wins l p w
      by simp
    thus False
      using condorcet-winner.simps l-in-A-without-w
            wins-antisym winner
      by metis
  qed
qed

theorem minimax-is-dcc: defer-condorcet-consistency minimax
proof −
  have max-mmaxscore-dcc:
    defer-condorcet-consistency (max-eliminator minimax-score)
    using cr-eval-imp-dcc-max-elim
    by (simp add: minimax-score-cond-rating)
  have mmax-eq-max-mmax:
    ⋀A p. (minimax A p ≡ max-eliminator minimax-score A p)
    by simp
  from max-mmaxscore-dcc mmax-eq-max-mmax
  show ?thesis
    unfolding defer-condorcet-consistency-def electoral-module-def
    by (smt (verit, ccfv-threshold))
qed

end
```

Chapter 4

Compositional Structures

4.1 Drop And Pass Compatibility

theory *Drop-And-Pass-Compatibility*
  imports *Basic-Modules/Drop-Module*
    Basic-Modules/Pass-Module
begin

This is a collection of properties about the interplay and compatibility of both the drop module and the pass module.

4.1.1 Properties

theorem *drop-two-mod-rej-two*[*simp*]:
  assumes *order*: *linear-order r*
  shows *rejects 2* (*drop-module 2 r*)
proof −
  have *reject* (*drop-module 2 r*) = *defer* (*pass-module 2 r*)
    by *simp*
  thus *?thesis*
  proof −
    obtain
      AA :: ('*a Electoral-Module*) ⇒ *nat* ⇒ '*a set* **and**
      rrs :: ('*a Electoral-Module*) ⇒ *nat* ⇒ '*a Profile* **where**
      ∀ *x0 x1*. (∃ *v2 v3*. (*x1* ≤ *card v2* ∧ *finite-profile v2 v3*) ∧
        card (*reject x0 v2 v3*) ≠ *x1*) =
          ((*x1* ≤ *card* (*AA x0 x1*) ∧ *finite-profile* (*AA x0 x1*) (*rrs x0 x1*)) ∧
          card (*reject x0* (*AA x0 x1*) (*rrs x0 x1*)) ≠ *x1*)
      by *moura*
    hence
      ∀ *n f*. (¬ *rejects n f* ∨ *electoral-module f* ∧
        (∀ *A rs*. (¬ *n* ≤ *card A* ∨ *infinite A* ∨ ¬ *profile A rs*) ∨
          card (*reject f A rs*) = *n*)) ∧
        (*rejects n f* ∨ ¬ *electoral-module f* ∨ (*n* ≤ *card* (*AA f n*) ∧
          finite-profile (*AA f n*) (*rrs f n*)) ∧

```
          card (reject f (AA f n) (rrs f n)) ≠ n)
      using rejects-def
      by force
    hence f1:
      ∀ n f. (¬ rejects n f ∨ electoral-module f ∧
        (∀ A rs. ¬ n ≤ card A ∨ infinite A ∨ ¬ profile A rs ∨
            card (reject f A rs) = n)) ∧
        (rejects n f ∨ ¬ electoral-module f ∨ n ≤ card (AA f n) ∧
            finite (AA f n) ∧ profile (AA f n) (rrs f n) ∧
            card (reject f (AA f n) (rrs f n)) ≠ n)
      by presburger
    have
      ¬ 2 ≤ card (AA (drop-module 2 r) 2) ∨
          infinite (AA (drop-module 2 r) 2) ∨
          ¬ profile (AA (drop-module 2 r) 2) (rrs (drop-module 2 r) 2) ∨
          card (reject (drop-module 2 r) (AA (drop-module 2 r) 2)
              (rrs (drop-module 2 r) 2)) = 2
      using ‹reject (drop-module 2 r) = defer (pass-module 2 r)›
            defers-def order pass-two-mod-def-two
      by (metis (no-types))
    thus ?thesis
      using f1 drop-mod-sound order
      by blast
  qed
qed


theorem drop-pass-disj-compat[simp]:
  assumes order: linear-order r
  shows disjoint-compatibility (drop-module n r) (pass-module n r)
  unfolding disjoint-compatibility-def
proof (safe)
  show electoral-module (drop-module n r)
    using order
    by simp
next
  show electoral-module (pass-module n r)
    using order
    by simp
next
  fix
    S :: 'a set
  assume
    fin: finite S
  obtain
    p :: 'a Profile
    where finite-profile S p
    using empty-iff empty-set fin profile-set
    by metis
```

```
  show
    ∃ A ⊆ S.
      (∀ a ∈ A. indep-of-alt (drop-module n r) S a ∧
        (∀ p. finite-profile S p ⟶
          a ∈ reject (drop-module n r) S p)) ∧
      (∀ a ∈ S−A. indep-of-alt (pass-module n r) S a ∧
        (∀ p. finite-profile S p ⟶
          a ∈ reject (pass-module n r) S p))
  proof
    have same-A:
      ∀ p q. (finite-profile S p ∧ finite-profile S q) ⟶
        reject (drop-module n r) S p =
          reject (drop-module n r) S q
      by auto
    let ?A = reject (drop-module n r) S p
    have ?A ⊆ S
      by auto
    moreover have
      (∀ a ∈ ?A. indep-of-alt (drop-module n r) S a)
      using order
      by (simp add: indep-of-alt-def)
    moreover have
      ∀ a ∈ ?A. ∀ p. finite-profile S p ⟶
        a ∈ reject (drop-module n r) S p
      by auto
    moreover have
      (∀ a ∈ S−?A. indep-of-alt (pass-module n r) S a)
      using order
      by (simp add: indep-of-alt-def)
    moreover have
      ∀ a ∈ S−?A. ∀ p. finite-profile S p ⟶
        a ∈ reject (pass-module n r) S p
      by auto
    ultimately show
      ?A ⊆ S ∧
        (∀ a ∈ ?A. indep-of-alt (drop-module n r) S a ∧
          (∀ p. finite-profile S p ⟶
            a ∈ reject (drop-module n r) S p)) ∧
        (∀ a ∈ S−?A. indep-of-alt (pass-module n r) S a ∧
          (∀ p. finite-profile S p ⟶
            a ∈ reject (pass-module n r) S p))
      by simp
  qed
qed

end
```

4.2 Revision Composition

theory *Revision-Composition*
 imports *Basic-Modules/Component-Types/Electoral-Module*
begin

A revised electoral module rejects all originally rejected or deferred alternatives, and defers the originally elected alternatives. It does not elect any alternatives.

4.2.1 Definition

fun *revision-composition* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* **where**
 revision-composition m A p = ({}, *A* − *elect m A p*, *elect m A p*)

abbreviation *rev* ::
'a Electoral-Module ⇒ *'a Electoral-Module* (-↓ *50*) **where**
 m↓ == *revision-composition m*

4.2.2 Soundness

theorem *rev-comp-sound*[*simp*]:
 assumes *module*: *electoral-module m*
 shows *electoral-module* (*revision-composition m*)
proof −
 from *module* **have** ∀ *A p*. *finite-profile A p* ⟶ *elect m A p* ⊆ *A*
  using *elect-in-alts*
  by *auto*
 hence ∀ *A p*. *finite-profile A p* ⟶ (*A* − *elect m A p*) ∪ *elect m A p* = *A*
  by *blast*
 hence *unity*:
  ∀ *A p*. *finite-profile A p* ⟶
   set-equals-partition A (*revision-composition m A p*)
  by *simp*
 have ∀ *A p*. *finite-profile A p* ⟶ (*A* − *elect m A p*) ∩ *elect m A p* = {}
  by *blast*
 hence *disjoint*:
  ∀ *A p*. *finite-profile A p* ⟶ *disjoint3* (*revision-composition m A p*)
  by *simp*
 from *unity disjoint* **show** *?thesis*
  by (*simp add*: *electoral-modI*)
qed

4.2.3 Composition Rules

theorem *rev-comp-non-electing*[*simp*]:
 assumes *electoral-module m*

shows *non-electing* $(m\downarrow)$
by (*simp add*: *assms non-electing-def*)

theorem *rev-comp-non-blocking*[*simp*]:
assumes *electing m*
shows *non-blocking* $(m\downarrow)$
unfolding *non-blocking-def*
proof (*safe*, *simp-all*)
show *electoral-module* $(m\downarrow)$
using *assms electing-def rev-comp-sound*
by (*metis* (*no-types*, *lifting*))
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
x :: $'a$
assume
fin-A: *finite* A **and**
prof-A: *profile* A p **and**
no-elect: A − *elect* m A p = A **and**
x-in-A: $x \in A$
from *no-elect* **have** *non-elect*:
non-electing m
using *assms prof-A x-in-A fin-A electing-def empty-iff*
Diff-disjoint Int-absorb2 elect-in-alts
by (*metis* (*no-types*, *lifting*))
show *False*
using *non-elect assms electing-def empty-iff fin-A*
non-electing-def prof-A x-in-A
by (*metis* (*no-types*, *lifting*))
qed

theorem *rev-comp-def-inv-mono*[*simp*]:
assumes *invariant-monotonicity m*
shows *defer-invariant-monotonicity* $(m\downarrow)$
proof −
have ∀ A p q w. ($w \in$ *defer* $(m\downarrow)$ A p ∧ *lifted* A p q w) ⟶
(*defer* $(m\downarrow)$ A q = *defer* $(m\downarrow)$ A p ∨ *defer* $(m\downarrow)$ A q = $\{w\}$)
using *assms*
by (*simp add*: *invariant-monotonicity-def*)
moreover have *electoral-module* $(m\downarrow)$
using *assms rev-comp-sound invariant-monotonicity-def*
by *auto*
moreover have *non-electing* $(m\downarrow)$
using *assms rev-comp-non-electing invariant-monotonicity-def*
by *auto*
ultimately have *electoral-module* $(m\downarrow)$ ∧ *non-electing* $(m\downarrow)$ ∧

$(\forall A\ p\ q\ w.\ (w \in defer\ (m\downarrow)\ A\ p \wedge lifted\ A\ p\ q\ w) \longrightarrow$
$(defer\ (m\downarrow)\ A\ q = defer\ (m\downarrow)\ A\ p \vee defer\ (m\downarrow)\ A\ q = \{w\}))$
by *blast*
thus *?thesis*
using *defer-invariant-monotonicity-def*
by (*simp add*: *defer-invariant-monotonicity-def*)
qed

end

4.3 Sequential Composition

theory *Sequential-Composition*
imports *Basic-Modules/Component-Types/Electoral-Module*
begin

The sequential composition creates a new electoral module from two electoral modules. In a sequential composition, the second electoral module makes decisions over alternatives deferred by the first electoral module.

4.3.1 Definition

fun *sequential-composition* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* ⇒
'a Electoral-Module **where**
sequential-composition m n A p =
(*let new-A* = *defer m A p*;
new-p = *limit-profile new-A p in* (
(*elect m A p*) ∪ (*elect n new-A new-p*),
(*reject m A p*) ∪ (*reject n new-A new-p*),
defer n new-A new-p))

abbreviation *sequence* ::
'a Electoral-Module ⇒ *'a Electoral-Module* ⇒ *'a Electoral-Module*
(**infix** ▷ *50*) **where**
m ▷ *n* == *sequential-composition m n*

lemma *seq-comp-presv-disj*:
assumes *module-m*: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows *disjoint3* ((*m* ▷ *n*) *A p*)
proof −
let *?new-A* = *defer m A p*
let *?new-p* = *limit-profile ?new-A p*

from *module-m f-prof* **have** *disjoint-m*: *disjoint3* (*m A p*)
  using *electoral-module-def well-formed.simps*
  by *blast*
from *module-m module-n def-presv-fin-prof f-prof* **have** *disjoint-n*:
  (*disjoint3* (*n ?new-A ?new-p*))
  using *electoral-module-def well-formed.simps*
  by *metis*
with *disjoint-m module-m module-n f-prof* **have** *0*:
  (*elect m A p* ∩ *reject n ?new-A ?new-p*) = {}
  using *disjoint-iff-not-equal reject-in-alts*
    def-presv-fin-prof result-disj subset-eq
  by (*smt* (*verit*, *best*))
from *disjoint-m disjoint-n def-presv-fin-prof f-prof*
    module-m module-n **have** *1*:
  (*elect m A p* ∩ *defer n ?new-A ?new-p*) = {}
  using *defer-in-alts disjoint-iff-not-equal*
    rev-subsetD result-disj distrib-imp2
    Int-Un-distrib inf-sup-distrib1
    result-presv-alts sup-bot.left-neutral
    sup-bot.neutr-eq-iff sup-bot-right 0
  by (*smt* (*verit*, *del-insts*))
from *disjoint-m disjoint-n def-presv-fin-prof f-prof*
    module-m module-n **have** *2*:
  (*reject m A p* ∩ *reject n ?new-A ?new-p*) = {}
  using *disjoint-iff-not-equal reject-in-alts*
    set-rev-mp result-disj Int-Un-distrib2
    Un-Diff-Int boolean-algebra-cancel.inf2
    inf.order-iff inf-sup-aci(*1*) *subsetD*
  by (*smt* (*verit*, *ccfv-threshold*))
from *disjoint-m disjoint-n def-presv-fin-prof f-prof*
    module-m module-n **have** *3*:
  (*reject m A p* ∩ *elect n ?new-A ?new-p*) = {}
  using *disjoint-iff-not-equal elect-in-alts set-rev-mp*
    result-disj Int-commute boolean-algebra-cancel.inf2
    defer-not-elec-or-rej inf.commute inf.orderE inf-commute
  by (*smt* (*verit*, *ccfv-threshold*))
from *0 1 2 3 disjoint-m disjoint-n module-m module-n f-prof* **have**
  (*elect m A p* ∪ *elect n ?new-A ?new-p*) ∩
    (*reject m A p* ∪ *reject n ?new-A ?new-p*) = {}
  using *inf-sup-aci*(*1*) *inf-sup-distrib2 def-presv-fin-prof*
    result-disj sup-inf-absorb sup-inf-distrib1
    distrib(*3*) *sup-eq-bot-iff*
  by (*smt* (*verit*, *ccfv-threshold*))
moreover from *0 1 2 3 disjoint-n module-m module-n f-prof* **have**
  (*elect m A p* ∪ *elect n ?new-A ?new-p*) ∩
    (*defer n ?new-A ?new-p*) = {}
  using *Int-Un-distrib2 Un-empty def-presv-fin-prof result-disj*
  by *metis*
moreover from *0 1 2 3 f-prof disjoint-m disjoint-n module-m module-n*

have
  (*reject m A p* ∪ *reject n ?new-A ?new-p*) ∩
      (*defer n ?new-A ?new-p*) = {}
  using *Int-Un-distrib2 defer-in-alts distrib-imp2*
      def-presv-fin-prof result-disj subset-Un-eq
      sup-inf-distrib1
  by (*smt* (*verit*))
ultimately have
  disjoint3 (*elect m A p* ∪ *elect n ?new-A ?new-p*,
      reject m A p ∪ *reject n ?new-A ?new-p*,
      defer n ?new-A ?new-p)
  by *simp*
thus *?thesis*
  using *sequential-composition.simps*
  by *metis*
qed

lemma *seq-comp-presv-alts*:
assumes *module-m*: *electoral-module m* **and**
    module-n: *electoral-module n* **and**
    f-prof: *finite-profile A p*
shows *set-equals-partition A* ((*m* ▷ *n*) *A p*)
proof −
let *?new-A* = *defer m A p*
let *?new-p* = *limit-profile ?new-A p*
from *module-m f-prof* **have** *set-equals-partition A* (*m A p*)
  by (*simp add*: *electoral-module-def*)
with *module-m f-prof* **have** *0*:
  elect m A p ∪ *reject m A p* ∪ *?new-A* = *A*
  by (*simp add*: *result-presv-alts*)
from *module-n def-presv-fin-prof f-prof module-m* **have**
  set-equals-partition ?new-A (*n ?new-A ?new-p*)
  using *electoral-module-def well-formed.simps*
  by *metis*
with *module-m module-n f-prof* **have** *1*:
  elect n ?new-A ?new-p ∪
    reject n ?new-A ?new-p ∪
    defer n ?new-A ?new-p = *?new-A*
  using *def-presv-fin-prof result-presv-alts*
  by *metis*
from *0 1* **have**
  (*elect m A p* ∪ *elect n ?new-A ?new-p*) ∪
    (*reject m A p* ∪ *reject n ?new-A ?new-p*) ∪
    defer n ?new-A ?new-p = *A*
  by *blast*
hence
  set-equals-partition A
    (*elect m A p* ∪ *elect n ?new-A ?new-p*,
    reject m A p ∪ *reject n ?new-A ?new-p*,

defer n ?new-A ?new-p)
by *simp*
thus *?thesis*
using *sequential-composition.simps*
by *metis*
qed

4.3.2 Soundness

theorem *seq-comp-sound*[*simp*]:
assumes *module-m*: *electoral-module m* **and**
module-n: *electoral-module n*
shows *electoral-module* (*m* ▷ *n*)
unfolding *electoral-module-def*
proof (*safe*)
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
fin-A: *finite A* **and**
prof-A: *profile A p*
have ∀ *r*. *well-formed* (*A*::*'a set*) *r* =
(*disjoint3 r* ∧ *set-equals-partition A r*)
by *simp*
thus *well-formed A* ((*m* ▷ *n*) *A p*)
using *module-m module-n seq-comp-presv-disj*
seq-comp-presv-alts fin-A prof-A
by *metis*
qed

4.3.3 Lemmata

lemma *seq-comp-dec-only-def*:
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p* **and**
empty-defer: *defer m A p* = {}
shows (*m* ▷ *n*) *A p* = *m A p*
using *Int-lower1 Un-absorb2 bot.extremum-uniqueI defer-in-alts*
elect-in-alts empty-defer module-m module-n prod.collapse
f-prof reject-in-alts sequential-composition.simps
def-presv-fin-prof result-disj
by (*smt* (*verit*))

lemma *seq-comp-def-then-elect*:
assumes
n-electing-m: *non-electing m* **and**
def-one-m: *defers 1 m* **and**
electing-n: *electing n* **and**

f-prof: *finite-profile A p*
shows *elect* $(m \triangleright n)$ *A p* = *defer m A p*
proof *cases*
assume $A = \{\}$
with *electing-n n-electing-m f-prof* **show** *?thesis*
using *bot.extremum-uniqueI defer-in-alts elect-in-alts electing-def non-electing-def seq-comp-sound*
by *metis*
next
assume *assm*: $A \neq \{\}$
from *n-electing-m f-prof* **have** *ele*: *elect m A p* = $\{\}$
using *non-electing-def*
by *auto*
from *assm def-one-m f-prof finite* **have** *def-card*:
card (*defer m A p*) = *1*
by (*simp add*: *Suc-leI card-gt-0-iff defers-def*)
with *n-electing-m f-prof* **have** *def*:
$\exists\, a \in A$. *defer m A p* = $\{a\}$
using *card-1-singletonE defer-in-alts non-electing-def singletonI subsetCE*
by *metis*
from *ele def n-electing-m* **have** *rej*:
$\exists\, a \in A$. *reject m A p* = $A-\{a\}$
using *Diff-empty def-one-m defers-def f-prof reject-not-elec-or-def*
by *metis*
from *ele rej def n-electing-m f-prof* **have** *res-m*:
$\exists\, a \in A$. *m A p* = $(\{\}, A-\{a\}, \{a\})$
using *Diff-empty combine-ele-rej-def non-electing-def reject-not-elec-or-def*
by *metis*
hence
$\exists\, a \in A$. *elect* $(m \triangleright n)$ *A p* =
elect n $\{a\}$ (*limit-profile* $\{a\}$ *p*)
using *prod.sel*(*1*) *prod.sel*(*2*) *sequential-composition.simps sup-bot.left-neutral*
by *metis*
with *def-card def electing-n n-electing-m f-prof* **have**
$\exists\, a \in A$. *elect* $(m \triangleright n)$ *A p* = $\{a\}$
using *electing-for-only-alt non-electing-def prod.sel sequential-composition.simps def-presv-fin-prof sup-bot.left-neutral*
by *metis*
with *def def-card electing-n n-electing-m f-prof res-m*
show *?thesis*
using *Diff-disjoint Diff-insert-absorb Int-insert-right Un-Diff-Int electing-for-only-alt empty-iff non-electing-def prod.sel sequential-composition.simps def-presv-fin-prof singletonI f-prof*
by (*smt* (*verit*, *best*))

qed

lemma *seq-comp-def-card-bounded*:
  assumes
    module-m: *electoral-module* m **and**
    module-n: *electoral-module* n **and**
    f-prof: *finite-profile* A p
  shows *card* (*defer* ($m \triangleright n$) A p) $\leq$ *card* (*defer* m A p)
  using *card-mono defer-in-alts module-m module-n f-prof*
      sequential-composition.simps def-presv-fin-prof snd-conv
  by *metis*

lemma *seq-comp-def-set-bounded*:
  assumes
    module-m: *electoral-module* m **and**
    module-n: *electoral-module* n **and**
    f-prof: *finite-profile* A p
  shows *defer* ($m \triangleright n$) A p $\subseteq$ *defer* m A p
  using *defer-in-alts module-m module-n prod.sel*(*2*) *f-prof*
      sequential-composition.simps def-presv-fin-prof
  by *metis*

lemma *seq-comp-defers-def-set*:
  assumes
    module-m: *electoral-module* m **and**
    module-n: *electoral-module* n **and**
    f-prof: *finite-profile* A p
  shows
    defer ($m \triangleright n$) A p =
      defer n (*defer* m A p) (*limit-profile* (*defer* m A p) p)
  using *sequential-composition.simps snd-conv*
  by *metis*

lemma *seq-comp-def-then-elect-elec-set*:
  assumes
    module-m: *electoral-module* m **and**
    module-n: *electoral-module* n **and**
    f-prof: *finite-profile* A p
  shows
    elect ($m \triangleright n$) A p =
      elect n (*defer* m A p) (*limit-profile* (*defer* m A p) p) $\cup$
      (*elect* m A p)
  using *Un-commute fst-conv sequential-composition.simps*
  by *metis*

lemma *seq-comp-elim-one-red-def-set*:
  assumes
    module-m: *electoral-module* m **and**
    module-n: *eliminates 1* n **and**

f-prof: *finite-profile A p* **and**
enough-leftover: *card* (*defer m A p*) > *1*
shows *defer* ($m \triangleright n$) *A p* ⊂ *defer m A p*
using *enough-leftover module-m module-n f-prof*
sequential-composition.simps def-presv-fin-prof
single-elim-imp-red-def-set snd-conv
by *metis*

lemma *seq-comp-def-set-sound*:
assumes
electoral-module m **and**
electoral-module n **and**
finite-profile A p
shows *defer* ($m \triangleright n$) *A p* ⊆ *defer m A p*
proof −
have ∀ *A p*. *finite-profile A p* ⟶ *well-formed A* (*n A p*)
using *assms*(*2*) *electoral-module-def*
by *auto*
hence
finite-profile (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) ⟶
well-formed (*defer m A p*)
(*n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*))
by *simp*
hence
well-formed (*defer m A p*) (*n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*))
using *assms*(*1*) *assms*(*3*) *def-presv-fin-prof*
by *metis*
thus *?thesis*
using *assms seq-comp-def-set-bounded*
by *blast*
qed

lemma *seq-comp-def-set-trans*:
assumes
a ∈ (*defer* ($m \triangleright n$) *A p*) **and**
electoral-module m ∧ *electoral-module n* **and**
finite-profile A p
shows
a ∈ *defer n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*) ∧
a ∈ *defer m A p*
using *seq-comp-def-set-bounded assms*(*1*) *assms*(*2*)
assms(*3*) *in-mono seq-comp-defers-def-set*
by (*metis* (*no-types*, *hide-lams*))

4.3.4 Composition Rules

theorem *seq-comp-presv-non-blocking*[*simp*]:

assumes
non-blocking-m: *non-blocking m* **and**
non-blocking-n: *non-blocking n*
shows *non-blocking* $(m \triangleright n)$
proof $-$
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
let *?input-sound* $= ((A{::}'a\ set) \neq \{\} \wedge$ *finite-profile* $A\ p)$
from *non-blocking-m* **have**
?input-sound $\longrightarrow$ *reject* $m\ A\ p \neq A$
by (*simp add*: *non-blocking-def*)
with *non-blocking-m* **have** *0*:
?input-sound $\longrightarrow A -$ *reject* $m\ A\ p \neq \{\}$
using *Diff-eq-empty-iff non-blocking-def*
reject-in-alts subset-antisym
by *metis*
from *non-blocking-m* **have**
?input-sound $\longrightarrow$ *well-formed* $A\ (m\ A\ p)$
by (*simp add*: *electoral-module-def non-blocking-def*)
hence
?input-sound $\longrightarrow$
elect $m\ A\ p \cup$ *defer* $m\ A\ p = A -$ *reject* $m\ A\ p$
using *non-blocking-def non-blocking-m elec-and-def-not-rej*
by *metis*
with *0* **have**
?input-sound $\longrightarrow$ *elect* $m\ A\ p \cup$ *defer* $m\ A\ p \neq \{\}$
by *auto*
hence *?input-sound* $\longrightarrow$ (*elect* $m\ A\ p \neq \{\} \vee$ *defer* $m\ A\ p \neq \{\}$)
by *simp*
with *non-blocking-m non-blocking-n*
show *?thesis*
using *Diff-empty Diff-subset-conv Un-empty fst-conv snd-conv*
defer-not-elec-or-rej elect-in-alts inf.absorb1 sup-bot-right
non-blocking-def reject-in-alts sequential-composition.simps
seq-comp-sound def-presv-fin-prof result-disj subset-antisym
by (*smt* (*verit*))
qed

theorem *seq-comp-presv-non-electing*[*simp*]:
assumes
non-electing m **and**
non-electing n
shows *non-electing* $(m \triangleright n)$
using *Un-empty assms non-electing-def prod.sel seq-comp-sound*
sequential-composition.simps def-presv-fin-prof
by (*smt* (*verit*, *del-insts*))

theorem *seq-comp-electing*[*simp*]:
  assumes *def-one-m1*: *defers 1 m1* **and**
    electing-m2: *electing m2*
  shows *electing* (*m1* ▷ *m2*)
proof –
  have
    ∀ *A p*. (*card A* ≥ *1* ∧ *finite-profile A p*) ⟶
      card (*defer m1 A p*) = *1*
    using *def-one-m1 defers-def*
    by *blast*
  hence
    ∀ *A p*. (*A* ≠ {} ∧ *finite-profile A p*) ⟶
      defer m1 A p ≠ {}
    using *One-nat-def Suc-leI card-eq-0-iff*
      card-gt-0-iff zero-neq-one
    by *metis*
  thus *?thesis*
    using *Un-empty def-one-m1 defers-def electing-def*
      electing-m2 seq-comp-def-then-elect-elec-set
      seq-comp-sound def-presv-fin-prof
    by (*smt* (*verit*, *ccfv-threshold*))
qed

lemma *def-lift-inv-seq-comp-help*:
  assumes
    monotone-m: *defer-lift-invariance m* **and**
    monotone-n: *defer-lift-invariance n* **and**
    def-and-lifted: *a* ∈ (*defer* (*m* ▷ *n*) *A p*) ∧ *lifted A p q a*
  shows (*m* ▷ *n*) *A p* = (*m* ▷ *n*) *A q*
proof –
  let *?new-Ap* = *defer m A p*
  let *?new-Aq* = *defer m A q*
  let *?new-p* = *limit-profile ?new-Ap p*
  let *?new-q* = *limit-profile ?new-Aq q*
  from *monotone-m monotone-n* **have** *modules*:
    electoral-module m ∧ *electoral-module n*
    by (*simp add*: *defer-lift-invariance-def*)
  hence *finite-profile A p* ⟶ *defer* (*m* ▷ *n*) *A p* ⊆ *defer m A p*
    using *seq-comp-def-set-bounded*
    by *metis*
  moreover have *profile-p*: *lifted A p q a* ⟶ *finite-profile A p*
    by (*simp add*: *lifted-def*)
  ultimately have *defer-subset*: *defer* (*m* ▷ *n*) *A p* ⊆ *defer m A p*
    using *def-and-lifted*
    by *blast*
  hence *mono-m*: *m A p* = *m A q*
    using *monotone-m defer-lift-invariance-def def-and-lifted*
      modules profile-p seq-comp-def-set-trans

```
  by metis
hence new-A-eq: ?new-Ap = ?new-Aq
  by presburger
have defer-eq:
  defer (m ▷ n) A p = defer n ?new-Ap ?new-p
  using sequential-composition.simps snd-conv
  by metis
hence mono-n:
  n ?new-Ap ?new-p = n ?new-Aq ?new-q
proof cases
  assume lifted ?new-Ap ?new-p ?new-q a
  thus ?thesis
    using defer-eq mono-m monotone-n
          defer-lift-invariance-def def-and-lifted
    by (metis (no-types, lifting))
next
  assume a2: ¬lifted ?new-Ap ?new-p ?new-q a
  from def-and-lifted have finite-profile A q
    by (simp add: lifted-def)
  with modules new-A-eq have 1:
    finite-profile ?new-Ap ?new-q
    using def-presv-fin-prof
    by (metis (no-types))
  moreover from modules profile-p def-and-lifted
  have 0:
    finite-profile ?new-Ap ?new-p
    using def-presv-fin-prof
    by (metis (no-types))
  moreover from defer-subset def-and-lifted
  have 2: a ∈ ?new-Ap
    by blast
  moreover from def-and-lifted have eql-lengths:
    size ?new-p = size ?new-q
    by (simp add: lifted-def)
  ultimately have 0:
    (∀ i::nat. i < size ?new-p ⟶
        ¬Preference-Relation.lifted ?new-Ap (?new-p!i) (?new-q!i) a) ∨
     (∃ i::nat. i < size ?new-p ∧
        ¬Preference-Relation.lifted ?new-Ap (?new-p!i) (?new-q!i) a ∧
          (?new-p!i) ≠ (?new-q!i))
    using a2 lifted-def
    by (metis (no-types, lifting))
  from def-and-lifted modules have
    ∀ i. (0 ≤ i ∧ i < size ?new-p) ⟶
        (Preference-Relation.lifted A (p!i) (q!i) a ∨ (p!i) = (q!i))
    using defer-in-alts Profile.lifted-def limit-prof-presv-size
    by metis
  with def-and-lifted modules mono-m have
    ∀ i. (0 ≤ i ∧ i < size ?new-p) ⟶
```

(*Preference-Relation.lifted ?new-Ap* (*?new-p*!*i*) (*?new-q*!*i*) *a* ∨
(*?new-p*!*i*) = (*?new-q*!*i*))
using *limit-lifted-imp-eq-or-lifted defer-in-alts*
Profile.lifted-def limit-prof-presv-size
limit-profile.simps nth-map
by (*metis* (*no-types*, *lifting*))
with *0 eql-lengths mono-m*
show *?thesis*
using *leI not-less-zero nth-equalityI*
by *metis*
qed
from *mono-m mono-n*
show *?thesis*
using *sequential-composition.simps*
by (*metis* (*full-types*))
qed

theorem *seq-comp-presv-def-lift-inv*[*simp*]:
assumes
monotone-m: *defer-lift-invariance m* **and**
monotone-n: *defer-lift-invariance n*
shows *defer-lift-invariance* ($m \triangleright n$)
using *monotone-m monotone-n def-lift-inv-seq-comp-help*
seq-comp-sound defer-lift-invariance-def
by (*metis* (*full-types*))

theorem *seq-comp-def-one*[*simp*]:
assumes
non-blocking-m: *non-blocking m* **and**
non-electing-m: *non-electing m* **and**
def-1-n: *defers 1 n*
shows *defers 1* ($m \triangleright n$)
unfolding *defers-def*
proof (*safe*)
have *electoral-mod-m*: *electoral-module m*
using *non-electing-m*
by (*simp add*: *non-electing-def*)
have *electoral-mod-n*: *electoral-module n*
using *def-1-n*
by (*simp add*: *defers-def*)
show *electoral-module* ($m \triangleright n$)
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile*

assume
  pos-card: $1 \leq$ *card A* **and**
  fin-A: *finite A* **and**
  prof-A: *profile A p*
from *pos-card* **have**
  $A \neq \{\}$
  by *auto*
with *fin-A prof-A* **have** *m-non-blocking*:
  reject m A p $\neq A$
  using *non-blocking-m non-blocking-def*
  by *metis*
hence
  $\exists\, a.\ a \in A \wedge a \notin$ *reject m A p*
  using *pos-card non-electing-def non-electing-m*
    reject-in-alts subset-antisym subset-iff
    fin-A prof-A subsetI
  by *metis*
hence *defer m A p* $\neq \{\}$
  using *electoral-mod-defer-elem empty-iff pos-card*
    non-electing-def non-electing-m fin-A prof-A
  by (*metis* (*no-types*))
hence *defer-non-empty*:
  card (*defer m A p*) ≥ 1
  using *One-nat-def Suc-leI card-gt-0-iff pos-card fin-A prof-A*
    non-blocking-def non-blocking-m def-presv-fin-prof
  by *metis*
have *defer-fun*:
  defer $(m \triangleright n)$ *A p* =
    defer n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
  using *def-1-n defers-def fin-A non-blocking-def non-blocking-m*
    prof-A seq-comp-defers-def-set
  by (*metis* (*no-types*, *hide-lams*))
have
  $\forall\, n\, f.$ *defers n f* =
    (*electoral-module f* $\wedge$
      ($\forall$ *A rs.*
        ($\neg\ n \leq$ *card* (*A*::*'a set*) $\vee$ *infinite A* $\vee$
          $\neg$ *profile A rs*) $\vee$
        card (*defer f A rs*) = *n*))
  using *defers-def*
  by *blast*
hence
  card (*defer n* (*defer m A p*)
    (*limit-profile* (*defer m A p*) *p*)) = *1*
  using *defer-non-empty def-1-n*
    fin-A prof-A non-blocking-def
    non-blocking-m def-presv-fin-prof
  by *metis*
thus *card* (*defer* $(m \triangleright n)$ *A p*) = *1*

using *defer-fun*
by *auto*
qed

theorem *disj-compat-seq*[*simp*]:
assumes
compatible: *disjoint-compatibility m n* **and**
module-m2: *electoral-module m2*
shows *disjoint-compatibility* (*m* ▷ *m2*) *n*
unfolding *disjoint-compatibility-def*
proof (*safe*)
show *electoral-module* (*m* ▷ *m2*)
using *compatible disjoint-compatibility-def module-m2 seq-comp-sound*
by *metis*
next
show *electoral-module n*
using *compatible disjoint-compatibility-def*
by *metis*
next
fix
S :: *'a set*
assume
fin-S: *finite S*
have *modules*:
electoral-module (*m* ▷ *m2*) ∧ *electoral-module n*
using *compatible disjoint-compatibility-def module-m2 seq-comp-sound*
by *metis*
obtain *A* **where** *A*:
$A \subseteq S \wedge$
$(\forall\, a \in A.$ *indep-of-alt m S a* $\wedge$
$(\forall\, p.$ *finite-profile S p* $\longrightarrow a \in$ *reject m S p*$)) \wedge$
$(\forall\, a \in S{-}A.$ *indep-of-alt n S a* $\wedge$
$(\forall\, p.$ *finite-profile S p* $\longrightarrow a \in$ *reject n S p*$))$
using *compatible disjoint-compatibility-def fin-S*
by (*metis* (*no-types*, *lifting*))
show
$\exists\, A \subseteq S.$
$(\forall\, a \in A.$ *indep-of-alt* (*m* ▷ *m2*) *S a* $\wedge$
$(\forall\, p.$ *finite-profile S p* $\longrightarrow a \in$ *reject* (*m* ▷ *m2*) *S p*$)) \wedge$
$(\forall\, a \in S{-}A.$ *indep-of-alt n S a* $\wedge$
$(\forall\, p.$ *finite-profile S p* $\longrightarrow a \in$ *reject n S p*$))$
proof
have
$\forall\, a\ p\ q.$
$a \in A \wedge$ *equiv-prof-except-a S p q a* $\longrightarrow$
(*m* ▷ *m2*) *S p* = (*m* ▷ *m2*) *S q*
proof (*safe*)
fix

a :: $'a$ **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile*
assume
a: $a \in A$ **and**
b: *equiv-prof-except-a S p q a*
have *eq-def*:
defer m S p = *defer m S q*
using *A a b indep-of-alt-def*
by *metis*
from *a b* **have** *profiles*:
finite-profile S p ∧ *finite-profile S q*
using *equiv-prof-except-a-def*
by *fastforce*
hence (*defer m S p*) ⊆ *S*
using *compatible defer-in-alts disjoint-compatibility-def*
by *blast*
hence
limit-profile (*defer m S p*) *p* =
limit-profile (*defer m S q*) *q*
using *A DiffD2 a b compatible defer-not-elec-or-rej*
disjoint-compatibility-def eq-def profiles
negl-diff-imp-eq-limit-prof
by (*metis* (*no-types*, *lifting*))
with *eq-def* **have**
m2 (*defer m S p*) (*limit-profile* (*defer m S p*) *p*) =
m2 (*defer m S q*) (*limit-profile* (*defer m S q*) *q*)
by *simp*
moreover have *m S p* = *m S q*
using *A a b indep-of-alt-def*
by *metis*
ultimately show
($m \triangleright m2$) *S p* = ($m \triangleright m2$) *S q*
using *sequential-composition.simps*
by (*metis* (*full-types*))
qed
moreover have
$\forall a \in A.\ \forall p.$ *finite-profile S p* ⟶ $a \in$ *reject* ($m \triangleright m2$) *S p*
using *A UnI1 prod.sel sequential-composition.simps*
by *metis*
ultimately show
$A \subseteq S$ ∧
($\forall a \in A.$ *indep-of-alt* ($m \triangleright m2$) *S a* ∧
($\forall p.$ *finite-profile S p* ⟶ $a \in$ *reject* ($m \triangleright m2$) *S p*)) ∧
($\forall a \in S-A.$ *indep-of-alt n S a* ∧
($\forall p.$ *finite-profile S p* ⟶ $a \in$ *reject n S p*))
using *A indep-of-alt-def modules*
by (*metis* (*mono-tags*, *lifting*))
qed

qed

theorem *seq-comp-mono*[*simp*]:
  assumes
    def-monotone-m: *defer-lift-invariance* m **and**
    non-ele-m: *non-electing* m **and**
    def-one-m: *defers 1* m **and**
    electing-n: *electing* n
  shows *monotonicity* $(m \triangleright n)$
  unfolding *monotonicity-def*
proof (*safe*)
  have *electoral-mod-m*: *electoral-module* m
    using *non-ele-m*
    by (*simp add*: *non-electing-def*)
  have *electoral-mod-n*: *electoral-module* n
    using *electing-n*
    by (*simp add*: *electing-def*)
  show *electoral-module* $(m \triangleright n)$
    using *electoral-mod-m electoral-mod-n*
    by *simp*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    q :: $'a$ *Profile* **and**
    w :: $'a$
  assume
    fin-A: *finite* A **and**
    elect-w-in-p: $w \in$ *elect* $(m \triangleright n)$ A p **and**
    lifted-w: *Profile.lifted* A p q w
  have
    finite-profile A p $\wedge$ *finite-profile* A q
    using *lifted-w lifted-def*
    by *metis*
  thus $w \in$ *elect* $(m \triangleright n)$ A q
    using *seq-comp-def-then-elect defer-lift-invariance-def*
        elect-w-in-p lifted-w def-monotone-m non-ele-m
        def-one-m electing-n
    by *metis*
qed

theorem *def-inv-mono-imp-def-lift-inv*[*simp*]:
  assumes
    strong-def-mon-m: *defer-invariant-monotonicity* m **and**
    non-electing-n: *non-electing* n **and**
    defers-1: *defers 1* n **and**
    defer-monotone-n: *defer-monotonicity* n

shows *defer-lift-invariance* $(m \triangleright n)$
unfolding *defer-lift-invariance-def*
proof (*safe*)
have *electoral-mod-m*: *electoral-module m*
using *defer-invariant-monotonicity-def*
strong-def-mon-m
by *auto*
have *electoral-mod-n*: *electoral-module n*
using *defers-1 defers-def*
by *auto*
show *electoral-module* $(m \triangleright n)$
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile* **and**
a :: $'a$
assume
defer-a-p: $a \in$ *defer* $(m \triangleright n)$ A p **and**
lifted-a: *Profile.lifted* A p q a
from *strong-def-mon-m*
have *non-electing-m*: *non-electing m*
by (*simp add*: *defer-invariant-monotonicity-def*)
have *electoral-mod-m*: *electoral-module m*
using *strong-def-mon-m defer-invariant-monotonicity-def*
by *auto*
have *electoral-mod-n*: *electoral-module n*
using *defers-1 defers-def*
by *auto*
have *finite-profile-q*: *finite-profile* A q
using *lifted-a*
by (*simp add*: *Profile.lifted-def*)
have *finite-profile-p*: *profile* A p
using *lifted-a*
by (*simp add*: *Profile.lifted-def*)
show $(m \triangleright n)$ A p $=$ $(m \triangleright n)$ A q
proof *cases*
assume *not-unchanged*: *defer* m A $q \neq$ *defer* m A p
hence *a-single-defer*: $\{a\}$ $=$ *defer* m A q
using *strong-def-mon-m electoral-mod-n defer-a-p*
defer-invariant-monotonicity-def lifted-a
seq-comp-def-set-trans finite-profile-p
finite-profile-q
by *metis*
moreover have
$\{a\}$ $=$ *defer* m A $q \longrightarrow$ *defer* $(m \triangleright n)$ A $q \subseteq \{a\}$
using *finite-profile-q electoral-mod-m electoral-mod-n*

seq-comp-def-set-sound
by (*metis* (*no-types*, *hide-lams*))
ultimately have
($a \in$ *defer* m A p) $\longrightarrow$ *defer* ($m \triangleright n$) A $q \subseteq \{a\}$
by *blast*
moreover have
($a \in$ *defer* m A p) $\longrightarrow$ *card* (*defer* ($m \triangleright n$) A q) $=$ *1*
using *One-nat-def a-single-defer card-eq-0-iff*
card-insert-disjoint defers-1 defers-def
electoral-mod-m empty-iff finite.emptyI
seq-comp-defers-def-set order-refl
def-presv-fin-prof finite-profile-q
by *metis*
moreover have *defer-a-in-m-p*:
$a \in$ *defer* m A p
using *electoral-mod-m electoral-mod-n defer-a-p*
seq-comp-def-set-bounded finite-profile-p
finite-profile-q
by *blast*
ultimately have
defer ($m \triangleright n$) A $q = \{a\}$
using *Collect-mem-eq card-1-singletonE empty-Collect-eq*
insertCI subset-singletonD
by *metis*
moreover have
defer ($m \triangleright n$) A $p = \{a\}$
using *card-mono defers-def insert-subset Diff-insert-absorb*
seq-comp-def-set-bounded elect-in-alts non-electing-def
non-electing-n defers-1 One-nat-def card-0-eq empty-iff
card-1-singletonE card-Diff-singleton finite.emptyI
card-insert-disjoint def-presv-fin-prof defer-a-p
electoral-mod-m finite-Diff insertCI insert-Diff
finite-profile-p finite-profile-q seq-comp-defers-def-set
by (*smt* (*verit*))
ultimately have
defer ($m \triangleright n$) A $p =$ *defer* ($m \triangleright n$) A q
by *blast*
moreover have
elect ($m \triangleright n$) A $p =$ *elect* ($m \triangleright n$) A q
using *finite-profile-p finite-profile-q*
non-electing-m non-electing-n
seq-comp-presv-non-electing
non-electing-def
by *metis*
thus *?thesis*
using *calculation eq-def-and-elect-imp-eq*
electoral-mod-m electoral-mod-n
finite-profile-p seq-comp-sound
finite-profile-q

```
    by metis
next
  assume not-different-alternatives:
    ¬(defer m A q ≠ defer m A p)
  have elect m A p = {}
    using non-electing-m finite-profile-p finite-profile-q
    by (simp add: non-electing-def)
  moreover have elect m A q = {}
    using non-electing-m finite-profile-q
    by (simp add: non-electing-def)
  ultimately have elect-m-equal:
    elect m A p = elect m A q
    by simp
  from not-different-alternatives
  have same-alternatives: defer m A q = defer m A p
    by simp
  hence
    (limit-profile (defer m A p) p) =
      (limit-profile (defer m A p) q) ∨
        lifted (defer m A q)
          (limit-profile (defer m A p) p)
            (limit-profile (defer m A p) q) a
    using defer-in-alts electoral-mod-m
          lifted-a finite-profile-q
          limit-prof-eq-or-lifted
    by metis
  thus ?thesis
  proof
    assume
      limit-profile (defer m A p) p =
        limit-profile (defer m A p) q
    hence same-profile:
      limit-profile (defer m A p) p =
        limit-profile (defer m A q) q
      using same-alternatives
      by simp
    hence results-equal-n:
      n (defer m A q) (limit-profile (defer m A q) q) =
        n (defer m A p) (limit-profile (defer m A p) p)
      by (simp add: same-alternatives)
    moreover have results-equal-m: m A p = m A q
      using elect-m-equal same-alternatives
            finite-profile-p finite-profile-q
      by (simp add: electoral-mod-m eq-def-and-elect-imp-eq)
    hence (m ▷ n) A p = (m ▷ n) A q
      using same-profile
      by auto
    thus ?thesis
      by blast
```

next
  assume *still-lifted*:
    lifted (*defer m A q*) (*limit-profile* (*defer m A p*) *p*)
      (*limit-profile* (*defer m A p*) *q*) *a*
  hence *a-in-def-p*:
    $a \in$ *defer n* (*defer m A p*)
      (*limit-profile* (*defer m A p*) *p*)
    using *electoral-mod-m electoral-mod-n*
          finite-profile-p defer-a-p
          seq-comp-def-set-trans
          finite-profile-q
    by *metis*
  hence *a-still-deferred-p*:
    $\{a\} \subseteq$ *defer n* (*defer m A p*)
      (*limit-profile* (*defer m A p*) *p*)
    by *simp*
  have *card-le-1-p*: *card* (*defer m A p*) $\geq$ *1*
    using *One-nat-def Suc-leI card-gt-0-iff*
          electoral-mod-m electoral-mod-n
          equals0D finite-profile-p defer-a-p
          seq-comp-def-set-trans def-presv-fin-prof
          finite-profile-q
    by *metis*
  hence
    card (*defer n* (*defer m A p*)
      (*limit-profile* (*defer m A p*) *p*)) = *1*
    using *defers-1 defers-def electoral-mod-m*
          finite-profile-p def-presv-fin-prof
          finite-profile-q
    by *metis*
  hence *def-set-is-a-p*:
    $\{a\}$ = *defer n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
    using *a-still-deferred-p card-1-singletonE*
          insert-subset singletonD
    by *metis*
  have *a-still-deferred-q*:
    $a \in$ *defer n* (*defer m A q*)
      (*limit-profile* (*defer m A p*) *q*)
    using *still-lifted a-in-def-p*
          defer-monotonicity-def
          defer-monotone-n electoral-mod-m
          same-alternatives
          def-presv-fin-prof finite-profile-q
    by *metis*
  have *card* (*defer m A q*) $\geq$ *1*
    using *card-le-1-p same-alternatives*
    by *auto*
  hence
    card (*defer n* (*defer m A q*)

(*limit-profile* (*defer m A q*) *q*)) = *1*
using *defers-1 defers-def electoral-mod-m*
finite-profile-q def-presv-fin-prof
by *metis*
hence *def-set-is-a-q*:
{*a*} =
defer n (*defer m A q*)
(*limit-profile* (*defer m A q*) *q*)
using *a-still-deferred-q card-1-singletonE*
same-alternatives singletonD
by *metis*
have
defer n (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*) =
defer n (*defer m A q*)
(*limit-profile* (*defer m A q*) *q*)
using *def-set-is-a-q def-set-is-a-p*
by *auto*
thus *?thesis*
using *seq-comp-presv-non-electing*
eq-def-and-elect-imp-eq non-electing-def
finite-profile-p finite-profile-q
non-electing-m non-electing-n
seq-comp-defers-def-set
by *metis*
qed
qed
qed

end

4.4 Parallel Composition

theory *Parallel-Composition*
imports *Basic-Modules/Component-Types/Aggregator*
Basic-Modules/Component-Types/Electoral-Module
begin

The parallel composition composes a new electoral module from two electoral modules combined with an aggregator. Therein, the two modules each make a decision and the aggregator combines them to a single (aggregated) result.

4.4.1 Definition

fun *parallel-composition* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* ⇒

```
    'a Aggregator ⇒ 'a Electoral-Module where
  parallel-composition m n agg A p = agg A (m A p) (n A p)

abbreviation parallel :: 'a Electoral-Module ⇒ 'a Aggregator ⇒
    'a Electoral-Module ⇒ 'a Electoral-Module
      (- ‖_ - [50, 1000, 51] 50) where
  m ‖_a n == parallel-composition m n a
```

4.4.2 Soundness

```
theorem par-comp-sound[simp]:
  assumes
    mod-m: electoral-module m and
    mod-n: electoral-module n and
    agg-a: aggregator a
  shows electoral-module (m ‖_a n)
  unfolding electoral-module-def
proof (safe)
  fix
    A :: 'a set and
    p :: 'a Profile
  assume
    fin-A: finite A and
    prof-A: profile A p
  have well-formed A (a A (m A p) (n A p))
    using aggregator-def combine-ele-rej-def par-comp-result-sound
        electoral-module-def mod-m mod-n fin-A prof-A agg-a
    by (smt (verit, ccfv-threshold))
  thus well-formed A ((m ‖_a n) A p)
    by simp
qed
```

4.4.3 Composition Rule

```
theorem conserv-agg-presv-non-electing[simp]:
  assumes
    non-electing-m: non-electing m and
    non-electing-n: non-electing n and
    conservative: agg-conservative a
  shows non-electing (m ‖_a n)
  unfolding non-electing-def
proof (safe)
  have emod-m: electoral-module m
    using non-electing-m
    by (simp add: non-electing-def)
  have emod-n: electoral-module n
    using non-electing-n
    by (simp add: non-electing-def)
  have agg-a: aggregator a
    using conservative
```

    by (*simp add*: *agg-conservative-def*)
  thus *electoral-module* ($m \parallel_a n$)
    using *emod-m emod-n agg-a par-comp-sound*
    by *simp*
next
  fix
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    x :: *'a*
  assume
    fin-A: *finite A* **and**
    prof-A: *profile A p* **and**
    x-wins: $x \in$ *elect* ($m \parallel_a n$) *A p*
  have *emod-m*: *electoral-module m*
    using *non-electing-m*
    by (*simp add*: *non-electing-def*)
  have *emod-n*: *electoral-module n*
    using *non-electing-n*
    by (*simp add*: *non-electing-def*)
  have
    let c = (*a A* (*m A p*) (*n A p*)) *in*
      (*elect-r c* $\subseteq$ ((*elect m A p*) $\cup$ (*elect n A p*)))
    using *conservative agg-conservative-def*
        emod-m emod-n par-comp-result-sound
        combine-ele-rej-def fin-A prof-A
    by (*smt* (*verit*, *ccfv-SIG*))
  hence $x \in$ ((*elect m A p*) $\cup$ (*elect n A p*))
    using *x-wins*
    by *auto*
  thus $x \in \{\}$
    using *sup-bot-right non-electing-def fin-A*
        non-electing-m non-electing-n prof-A
    by (*metis* (*no-types*, *lifting*))
qed

end

4.5 Loop Composition

theory *Loop-Composition*
  imports *Basic-Modules/Component-Types/Termination-Condition*
        Basic-Modules/Defer-Module
        Sequential-Composition
begin

The loop composition uses the same module in sequence, combined with a termination condition, until either (1) the termination condition is met or (2) no new decisions are made (i.e., a fixed point is reached).

4.5.1 Definition

lemma *loop-termination-helper*:
  assumes
    not-term: $\neg t$ (*acc A p*) **and**
    subset: *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* **and**
    not-inf: $\neg$*infinite* (*defer acc A p*)
  shows
    ((*acc* $\triangleright$ *m*, *m*, *t*, *A*, *p*), (*acc*, *m*, *t*, *A*, *p*)) $\in$
      measure (λ(*acc*, *m*, *t*, *A*, *p*). *card* (*defer acc A p*))
  using *assms psubset-card-mono*
  by *auto*

function *loop-comp-helper* ::
    'a Electoral-Module $\Rightarrow$ *'a Electoral-Module* $\Rightarrow$
      'a Termination-Condition $\Rightarrow$ *'a Electoral-Module* **where**
  t (*acc A p*) $\vee$ $\neg$((*defer* (*acc* $\triangleright$ *m*) *A p*) $\subset$ (*defer acc A p*)) $\vee$
    infinite (*defer acc A p*) $\Longrightarrow$
      loop-comp-helper acc m t A p = *acc A p* |
  $\neg$(*t* (*acc A p*) $\vee$ $\neg$((*defer* (*acc* $\triangleright$ *m*) *A p*) $\subset$ (*defer acc A p*)) $\vee$
    infinite (*defer acc A p*)) $\Longrightarrow$
      loop-comp-helper acc m t A p = *loop-comp-helper* (*acc* $\triangleright$ *m*) *m t A p*
proof –
  fix
    P :: *bool* **and**
    x :: (*'a Electoral-Module*) $\times$ (*'a Electoral-Module*) $\times$
      (*'a Termination-Condition*) $\times$ *'a set* $\times$ *'a Profile*
  assume
    a1: $\bigwedge$*t acc A p m*.
      $[\![$*t* (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$
        infinite (*defer acc A p*);
        x = (*acc*, *m*, *t*, *A*, *p*)$]\!]$ $\Longrightarrow$ *P* **and**
    a2: $\bigwedge$*t acc A p m*.
      $[\![\neg$ (*t* (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$
        infinite (*defer acc A p*));
        x = (*acc*, *m*, *t*, *A*, *p*)$]\!]$ $\Longrightarrow$ *P*
  have $\exists$*f A p rs fa*. (*fa*, *f*, *p*, *A*, *rs*) = *x*
    using *prod-cases5*
    by *metis*
  then show *P*
    using *a2 a1*
    by (*metis* (*no-types*))
next
  show

```
    ⋀t acc A p m ta acca Aa pa ma.
       t (acc A p) ∨ ¬ defer (acc ▷ m) A p ⊂ defer acc A p ∨
        infinite (defer acc A p) ⟹
          ta (acca Aa pa) ∨ ¬ defer (acca ▷ ma) Aa pa ⊂ defer acca Aa pa ∨
          infinite (defer acca Aa pa) ⟹
           (acc, m, t, A, p) = (acca, ma, ta, Aa, pa) ⟹
              acc A p = acca Aa pa
    by fastforce
next
  show
    ⋀t acc A p m ta acca Aa pa ma.
       t (acc A p) ∨ ¬ defer (acc ▷ m) A p ⊂ defer acc A p ∨
        infinite (defer acc A p) ⟹
          ¬ (ta (acca Aa pa) ∨ ¬ defer (acca ▷ ma) Aa pa ⊂ defer acca Aa pa ∨
          infinite (defer acca Aa pa)) ⟹
           (acc, m, t, A, p) = (acca, ma, ta, Aa, pa) ⟹
              acc A p = loop-comp-helper-sumC (acca ▷ ma, ma, ta, Aa, pa)
  proof −
    fix
      t :: 'a Termination-Condition and
      acc :: 'a Electoral-Module and
      A :: 'a set and
      p :: 'a Profile and
      m :: 'a Electoral-Module and
      ta :: 'a Termination-Condition and
      acca :: 'a Electoral-Module and
      Aa :: 'a set and
      pa :: 'a Profile and
      ma :: 'a Electoral-Module
    assume
      a1: t (acc A p) ∨ ¬ defer (acc ▷ m) A p ⊂ defer acc A p ∨
            infinite (defer acc A p) and
      a2: ¬ (ta (acca Aa pa) ∨ ¬ defer (acca ▷ ma) Aa pa ⊂ defer acca Aa pa ∨
            infinite (defer acca Aa pa)) and
      (acc, m, t, A, p) = (acca, ma, ta, Aa, pa)
    hence False
      using a2 a1
      by force
  thus acc A p = loop-comp-helper-sumC (acca ▷ ma, ma, ta, Aa, pa)
    by auto
qed
next
  show
    ⋀t acc A p m ta acca Aa pa ma.
       ¬ (t (acc A p) ∨ ¬ defer (acc ▷ m) A p ⊂ defer acc A p ∨
          infinite (defer acc A p)) ⟹
            ¬ (ta (acca Aa pa) ∨ ¬ defer (acca ▷ ma) Aa pa ⊂ defer acca Aa pa ∨
            infinite (defer acca Aa pa)) ⟹
             (acc, m, t, A, p) = (acca, ma, ta, Aa, pa) ⟹
```

loop-comp-helper-sumC (*acc* ▷ *m*, *m*, *t*, *A*, *p*) =
  loop-comp-helper-sumC (*acca* ▷ *ma*, *ma*, *ta*, *Aa*, *pa*)
by *force*
qed
termination
proof −
have *f0*:
∃ *r*. *wf r* ∧
(∀ *p f A rs fa*.
p (*f* (*A*::′*a set*) *rs*) ∨
¬ *defer* (*f* ▷ *fa*) *A rs* ⊂ *defer f A rs* ∨
infinite (*defer f A rs*) ∨
((*f* ▷ *fa*, *fa*, *p*, *A*, *rs*), (*f*, *fa*, *p*, *A*, *rs*)) ∈ *r*)
using *loop-termination-helper wf-measure termination*
by (*metis* (*no-types*))
hence
∀ *r p*.
Ex ((λ*ra*. ∀ *f A rs pa fa*. ∃ *ra pb rb pc pd fb Aa rsa fc pe*.
¬ *wf r* ∨
loop-comp-helper-dom
(*p*::(′*a Electoral-Module*) × (- *Electoral-Module*) ×
(- *Termination-Condition*) × - *set* × - *Profile*) ∨
infinite (*defer f* (*A*::′*a set*) *rs*) ∨
pa (*f A rs*) ∧
wf
(*ra*::((
(′*a Electoral-Module*) × (′*a Electoral-Module*) ×
(′*a Termination-Condition*) × ′*a set* × ′*a Profile*) × -) *set*) ∧
¬ *loop-comp-helper-dom* (*pb*::
(′*a Electoral-Module*) × (- *Electoral-Module*) ×
(- *Termination-Condition*) × - *set* × - *Profile*) ∨
wf rb ∧ ¬ *defer* (*f* ▷ *fa*) *A rs* ⊂ *defer f A rs* ∧
¬ *loop-comp-helper-dom*
(*pc*::(′*a Electoral-Module*) × (- *Electoral-Module*) ×
(- *Termination-Condition*) × - *set* × - *Profile*) ∨
((*f* ▷ *fa*, *fa*, *pa*, *A*, *rs*), *f*, *fa*, *pa*, *A*, *rs*) ∈ *rb* ∧ *wf rb* ∧
¬ *loop-comp-helper-dom*
(*pd*::(′*a Electoral-Module*) × (- *Electoral-Module*) ×
(- *Termination-Condition*) × - *set* × - *Profile*) ∨
finite (*defer fb* (*Aa*::′*a set*) *rsa*) ∧
defer (*fb* ▷ *fc*) *Aa rsa* ⊂ *defer fb Aa rsa* ∧
¬ *pe* (*fb Aa rsa*) ∧
((*fb* ▷ *fc*, *fc*, *pe*, *Aa*, *rsa*), *fb*, *fc*, *pe*, *Aa*, *rsa*) ∉ *r*)::
(((′*a Electoral-Module*) × (′*a Electoral-Module*) ×
(′*a Termination-Condition*) × ′*a set* × ′*a Profile*) ×
(′*a Electoral-Module*) × (′*a Electoral-Module*) ×
(′*a Termination-Condition*) × ′*a set* × ′*a Profile*) *set* ⇒ *bool*)
by *metis*
obtain

rr :: $((('a$ *Electoral-Module*) × ('a *Electoral-Module*) ×
('a *Termination-Condition*) × 'a *set* × 'a *Profile*) ×
('a *Electoral-Module*) × ('a *Electoral-Module*) ×
('a *Termination-Condition*) × 'a *set* × 'a *Profile*) *set* **where**
wf rr ∧
(∀ *p f A rs fa*. *p* (*f A rs*) ∨
¬ *defer* (*f* ▷ *fa*) *A rs* ⊂ *defer f A rs* ∨
infinite (*defer f A rs*) ∨
((*f* ▷ *fa*, *fa*, *p*, *A*, *rs*), *f*, *fa*, *p*, *A*, *rs*) ∈ *rr*)
using *f0*
by *presburger*
thus *?thesis*
using *termination*
by *metis*
qed

lemma *loop-comp-code-helper*[*code*]:
loop-comp-helper acc m t A p =
(*if* (*t* (*acc A p*) ∨ ¬((*defer* (*acc* ▷ *m*) *A p*) ⊂ (*defer acc A p*)) ∨
infinite (*defer acc A p*))
then (*acc A p*) *else* (*loop-comp-helper* (*acc* ▷ *m*) *m t A p*))
by *simp*

function *loop-composition* ::
'a *Electoral-Module* ⇒ 'a *Termination-Condition* ⇒
'a *Electoral-Module* **where**
t ({}, {}, *A*) ⟹
loop-composition m t A p = *defer-module A p* |
¬(*t* ({}, {}, *A*)) ⟹
loop-composition m t A p = (*loop-comp-helper m m t*) *A p*
by (*fastforce*, *simp-all*)
termination
using *termination wf-empty*
by *blast*

abbreviation *loop* ::
'a *Electoral-Module* ⇒ 'a *Termination-Condition* ⇒ 'a *Electoral-Module*
(- $\circlearrowleft_{-}$ *50*) **where**
$m \circlearrowleft_t$ ≡ *loop-composition m t*

lemma *loop-comp-code*[*code*]:
loop-composition m t A p =
(*if* (*t* ({},{},*A*))
then (*defer-module A p*) *else* (*loop-comp-helper m m t*) *A p*)
by *simp*

lemma *loop-comp-helper-imp-partit*:
assumes
module-m: *electoral-module m* **and**

profile: *finite-profile A p*
shows
electoral-module acc ∧ (*n* = *card* (*defer acc A p*)) ⟹
well-formed A (*loop-comp-helper acc m t A p*)
proof (*induct arbitrary*: *acc rule*: *less-induct*)
case (*less*)
thus *?case*
using *electoral-module-def loop-comp-helper.simps*(*1*)
loop-comp-helper.simps(*2*) *module-m profile*
psubset-card-mono seq-comp-sound
by (*smt* (*verit*))
qed

4.5.2 Soundness

theorem *loop-comp-sound*:
assumes *m-module*: *electoral-module m*
shows *electoral-module* (m ↻$_t$)
using *def-mod-sound electoral-module-def loop-composition.simps*(*1*)
loop-composition.simps(*2*) *loop-comp-helper-imp-partit m-module*
by *metis*

lemma *loop-comp-helper-imp-no-def-incr*:
assumes
module-m: *electoral-module m* **and**
profile: *finite-profile A p*
shows
(*electoral-module acc* ∧ *n* = *card* (*defer acc A p*)) ⟹
defer (*loop-comp-helper acc m t*) *A p* ⊆ *defer acc A p*
proof (*induct arbitrary*: *acc rule*: *less-induct*)
case (*less*)
thus *?case*
using *dual-order.trans eq-iff less-imp-le loop-comp-helper.simps*(*1*)
loop-comp-helper.simps(*2*) *module-m psubset-card-mono*
seq-comp-sound
by (*smt* (*verit*, *ccfv-SIG*))
qed

4.5.3 Lemmata

lemma *loop-comp-helper-def-lift-inv-helper*:
assumes
monotone-m: *defer-lift-invariance m* **and**
f-prof: *finite-profile A p*
shows
(*defer-lift-invariance acc* ∧ *n* = *card* (*defer acc A p*)) ⟶
(∀ *q a*.
(*a* ∈ (*defer* (*loop-comp-helper acc m t*) *A p*) ∧
lifted A p q a) ⟶
(*loop-comp-helper acc m t*) *A p* =

```
            (loop-comp-helper acc m t) A q)
proof (induct n arbitrary: acc rule: less-induct)
  case (less n)
  have defer-card-comp:
    defer-lift-invariance acc ⟶
        (∀ q a. (a ∈ (defer (acc ▹ m) A p) ∧ lifted A p q a) ⟶
            card (defer (acc ▹ m) A p) = card (defer (acc ▹ m) A q))
    using monotone-m def-lift-inv-seq-comp-help
    by metis
  have defer-card-acc:
    defer-lift-invariance acc ⟶
        (∀ q a. (a ∈ (defer (acc) A p) ∧ lifted A p q a) ⟶
            card (defer (acc) A p) = card (defer (acc) A q))
    by (simp add: defer-lift-invariance-def)
  hence defer-card-acc-2:
    defer-lift-invariance acc ⟶
        (∀ q a. (a ∈ (defer (acc ▹ m) A p) ∧ lifted A p q a) ⟶
            card (defer (acc) A p) = card (defer (acc) A q))
    using monotone-m f-prof defer-lift-invariance-def seq-comp-def-set-trans
    by metis
  thus ?case
  proof cases
    assume card-unchanged: card (defer (acc ▹ m) A p) = card (defer acc A p)
    with defer-card-comp defer-card-acc monotone-m
    have
      defer-lift-invariance (acc) ⟶
          (∀ q a. (a ∈ (defer (acc) A p) ∧ lifted A p q a) ⟶
              (loop-comp-helper acc m t) A q = acc A q)
      using card-subset-eq defer-in-alts less-irrefl
            loop-comp-helper.simps(1) f-prof psubset-card-mono
            sequential-composition.simps def-presv-fin-prof snd-conv
            defer-lift-invariance-def seq-comp-def-set-bounded
            loop-comp-code-helper
      by (smt (verit))
    moreover from card-unchanged have
      (loop-comp-helper acc m t) A p = acc A p
      using loop-comp-helper.simps(1) order.strict-iff-order
            psubset-card-mono
      by metis
    ultimately have
      (defer-lift-invariance (acc ▹ m) ∧ defer-lift-invariance acc) ⟶
          (∀ q a. (a ∈ (defer (loop-comp-helper acc m t) A p) ∧
              lifted A p q a) ⟶
                  (loop-comp-helper acc m t) A p =
                    (loop-comp-helper acc m t) A q)
      using defer-lift-invariance-def
      by metis
    thus ?thesis
      using monotone-m seq-comp-presv-def-lift-inv
```

by *blast*
next
assume *card-changed*:
$\neg$ (*card* (*defer* (*acc* $\triangleright$ *m*) *A p*) = *card* (*defer acc A p*))
with *f-prof seq-comp-def-card-bounded* **have** *card-smaller-for-p*:
electoral-module (*acc*) $\longrightarrow$
(*card* (*defer* (*acc* $\triangleright$ *m*) *A p*) < *card* (*defer acc A p*))
using *monotone-m order.not-eq-order-implies-strict*
defer-lift-invariance-def
by (*metis* (*full-types*))
with *defer-card-acc-2 defer-card-comp* **have** *card-changed-for-q*:
defer-lift-invariance (*acc*) $\longrightarrow$
($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
(*card* (*defer* (*acc* $\triangleright$ *m*) *A q*) < *card* (*defer acc A q*)))
using *defer-lift-invariance-def*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
proof *cases*
assume *t-not-satisfied-for-p*: $\neg$ *t* (*acc A p*)
hence *t-not-satisfied-for-q*:
defer-lift-invariance (*acc*) $\longrightarrow$
($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
$\neg$ *t* (*acc A q*))
using *monotone-m f-prof defer-lift-invariance-def seq-comp-def-set-trans*
by *metis*
from *card-changed defer-card-comp defer-card-acc*
have
(*defer-lift-invariance* (*acc* $\triangleright$ *m*) $\wedge$ *defer-lift-invariance* (*acc*)) $\longrightarrow$
($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
card (*defer* (*acc* $\triangleright$ *m*) *A q*) $\neq$ (*card* (*defer acc A q*)))
proof –
have
$\forall$ *f*. (*defer-lift-invariance f* $\vee$
($\exists$ *A rs rsa a*. *f A rs* $\neq$ *f A rsa* $\wedge$
Profile.lifted A rs rsa (*a*::'*a*) $\wedge$
a $\in$ *defer f A rs*) $\vee$ $\neg$ *electoral-module f*) $\wedge$
(($\forall$ *A rs rsa a*. *f A rs* = *f A rsa* $\vee$ $\neg$ *Profile.lifted A rs rsa a* $\vee$
a $\notin$ *defer f A rs*) $\wedge$ *electoral-module f* $\vee$ $\neg$ *defer-lift-invariance f*)
using *defer-lift-invariance-def*
by *blast*
thus *?thesis*
using *card-changed monotone-m f-prof seq-comp-def-set-trans*
by (*metis* (*no-types*, *hide-lams*))
qed
hence
defer-lift-invariance (*acc* $\triangleright$ *m*) $\wedge$ *defer-lift-invariance* (*acc*) $\longrightarrow$
($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
defer (*acc* $\triangleright$ *m*) *A q* $\subset$ *defer acc A q*)
using *defer-card-acc defer-in-alts monotone-m prod.sel*(*2*) *f-prof*

```
            psubsetI sequential-composition.simps def-presv-fin-prof
            defer-lift-invariance-def subsetCE Profile.lifted-def
            seq-comp-def-set-bounded
        by (smt (verit))
      with t-not-satisfied-for-p have rec-step-q:
        (defer-lift-invariance (acc ▷ m) ∧ defer-lift-invariance (acc)) ⟶
            (∀ q a. (a ∈ (defer (acc ▷ m) A p) ∧ lifted A p q a) ⟶
                loop-comp-helper acc m t A q =
                  loop-comp-helper (acc ▷ m) m t A q)
        using defer-in-alts loop-comp-helper.simps(2) monotone-m subsetCE
            prod.sel(2) f-prof sequential-composition.simps card-eq-0-iff
            def-presv-fin-prof defer-lift-invariance-def card-changed-for-q
            gr-implies-not0 t-not-satisfied-for-q
        by (smt (verit, ccfv-SIG))
      have rec-step-p:
        electoral-module acc ⟶
            loop-comp-helper acc m t A p = loop-comp-helper (acc ▷ m) m t A p
        using card-changed defer-in-alts loop-comp-helper.simps(2)
            monotone-m prod.sel(2) f-prof psubsetI def-presv-fin-prof
            sequential-composition.simps defer-lift-invariance-def
            t-not-satisfied-for-p seq-comp-def-set-bounded
        by (smt (verit, best))
      thus ?thesis
        using card-smaller-for-p less.hyps
            loop-comp-helper-imp-no-def-incr monotone-m
            seq-comp-presv-def-lift-inv f-prof rec-step-q
            defer-lift-invariance-def subsetCE subset-eq
        by (smt (verit, ccfv-threshold))
    next
      assume t-satisfied-for-p: ¬ ¬t (acc A p)
      thus ?thesis
        using loop-comp-helper.simps(1) defer-lift-invariance-def
        by metis
    qed
  qed
qed

lemma loop-comp-helper-def-lift-inv:
  assumes
    monotone-m: defer-lift-invariance m and
    monotone-acc: defer-lift-invariance acc and
    profile: finite-profile A p
  shows
    ∀ q a. (lifted A p q a ∧ a ∈ (defer (loop-comp-helper acc m t) A p)) ⟶
        (loop-comp-helper acc m t) A p = (loop-comp-helper acc m t) A q
  using loop-comp-helper-def-lift-inv-helper
      monotone-m monotone-acc profile
  by blast
```

lemma *loop-comp-helper-def-lift-inv2*:
  assumes
    monotone-m: *defer-lift-invariance m* **and**
    monotone-acc: *defer-lift-invariance acc*
  shows
    $\forall$ *A p q a*. (*finite-profile A p* $\wedge$
      lifted A p q a $\wedge$
      a $\in$ (*defer* (*loop-comp-helper acc m t*) *A p*)) $\longrightarrow$
        (*loop-comp-helper acc m t*) *A p* = (*loop-comp-helper acc m t*) *A q*
  using *loop-comp-helper-def-lift-inv monotone-acc monotone-m*
  by *blast*

lemma *lifted-imp-fin-prof*:
  assumes *lifted A p q a*
  shows *finite-profile A p*
  using *assms Profile.lifted-def*
  by *fastforce*

lemma *loop-comp-helper-presv-def-lift-inv*:
  assumes
    monotone-m: *defer-lift-invariance m* **and**
    monotone-acc: *defer-lift-invariance acc*
  shows *defer-lift-invariance* (*loop-comp-helper acc m t*)
proof −
  have
    $\forall$*f*. (*defer-lift-invariance f* $\vee$
        ($\exists$ *A rs rsa a*. *f A rs* $\neq$ *f A rsa* $\wedge$
            Profile.lifted A rs rsa (*a*::'*a*) $\wedge$
            a $\in$ *defer f A rs*) $\vee$
        $\neg$ *electoral-module f*) $\wedge$
      (($\forall$ *A rs rsa a*. *f A rs* = *f A rsa* $\vee$ $\neg$ *Profile.lifted A rs rsa a* $\vee$
          a $\notin$ *defer f A rs*) $\wedge$
      electoral-module f $\vee$ $\neg$ *defer-lift-invariance f*)
    using *defer-lift-invariance-def*
    by *blast*
  thus *?thesis*
    using *electoral-module-def lifted-imp-fin-prof*
        loop-comp-helper-def-lift-inv loop-comp-helper-imp-partit
        monotone-acc monotone-m
    by (*metis* (*full-types*))
qed

lemma *loop-comp-presv-non-electing-helper*:
  assumes
    non-electing-m: *non-electing m* **and**
    f-prof: *finite-profile A p*
  shows
    (*n* = *card* (*defer acc A p*) $\wedge$ *non-electing acc*) $\Longrightarrow$
      elect (*loop-comp-helper acc m t*) *A p* = {}

proof (*induct n arbitrary*: *acc rule*: *less-induct*)
  case(*less n*)
  thus *?case*
    using *loop-comp-helper.simps*(*1*) *loop-comp-helper.simps*(*2*)
      non-electing-def non-electing-m f-prof psubset-card-mono
      seq-comp-presv-non-electing
    by (*smt* (*verit*, *ccfv-threshold*))
qed

lemma *loop-comp-helper-iter-elim-def-n-helper*:
  assumes
    non-electing-m: *non-electing m* **and**
    single-elimination: *eliminates 1 m* **and**
    terminate-if-n-left: $\forall$ *r*. ((*t r*) $\longleftrightarrow$ (*card* (*defer-r r*) = *x*)) **and**
    x-greater-zero: *x* > *0* **and**
    f-prof: *finite-profile A p*
  shows
    (*n* = *card* (*defer acc A p*) $\wedge$ *n* $\geq$ *x* $\wedge$ *card* (*defer acc A p*) > *1* $\wedge$
     non-electing acc) $\longrightarrow$
       card (*defer* (*loop-comp-helper acc m t*) *A p*) = *x*
proof (*induct n arbitrary*: *acc rule*: *less-induct*)
  case(*less n*)
  have *subset*:
    (*card* (*defer acc A p*) > *1* $\wedge$ *finite-profile A p* $\wedge$ *electoral-module acc*) $\longrightarrow$
      defer (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p*
    using *seq-comp-elim-one-red-def-set single-elimination*
    by *blast*
  hence *step-reduces-defer-set*:
    (*card* (*defer acc A p*) > *1* $\wedge$ *finite-profile A p* $\wedge$ *non-electing acc*) $\longrightarrow$
      defer (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p*
    using *non-electing-def*
    by *auto*
  thus *?case*
  proof *cases*
    assume *term-satisfied*: *t* (*acc A p*)
    have *card* (*defer-r* (*loop-comp-helper acc m t A p*)) = *x*
      using *loop-comp-helper.simps*(*1*) *term-satisfied terminate-if-n-left*
      by *metis*
    thus *?case*
      by *blast*
  next
    assume *term-not-satisfied*: $\neg$(*t* (*acc A p*))
    hence *card-not-eq-x*: *card* (*defer acc A p*) $\neq$ *x*
      by (*simp add*: *terminate-if-n-left*)
    have *rec-step*:
      (*card* (*defer acc A p*) > *1* $\wedge$ *finite-profile A p* $\wedge$ *non-electing acc*) $\longrightarrow$
        loop-comp-helper acc m t A p =
          loop-comp-helper (*acc* $\triangleright$ *m*) *m t A p*
      using *loop-comp-helper.simps*(*2*) *non-electing-def def-presv-fin-prof*

step-reduces-defer-set term-not-satisfied
by *metis*
thus *?case*
proof *cases*
assume *card-too-small*: *card* (*defer acc A p*) $<$ *x*
thus *?thesis*
using *not-le*
by *blast*
next
assume *old-card-at-least-x*: $\neg$(*card* (*defer acc A p*) $<$ *x*)
obtain *i* **where** *i-is-new-card*: *i* $=$ *card* (*defer* (*acc* $\triangleright$ *m*) *A p*)
by *blast*
with *card-not-eq-x* **have** *card-too-big*:
card (*defer acc A p*) $>$ *x*
using *nat-neq-iff old-card-at-least-x*
by *blast*
hence *enough-leftover*: *card* (*defer acc A p*) $>$ *1*
using *x-greater-zero*
by *auto*
have *electoral-module acc* $\longrightarrow$ (*defer acc A p*) $\subseteq$ *A*
by (*simp add*: *defer-in-alts f-prof*)
hence *step-profile*:
electoral-module acc $\longrightarrow$
finite-profile (*defer acc A p*)
(*limit-profile* (*defer acc A p*) *p*)
using *f-prof limit-profile-sound*
by *auto*
hence
electoral-module acc $\longrightarrow$
card (*defer m* (*defer acc A p*)
(*limit-profile* (*defer acc A p*) *p*)) $=$
card (*defer acc A p*) $-$ *1*
using *non-electing-m single-elimination*
single-elim-decr-def-card2 enough-leftover
by *blast*
hence *electoral-module acc* $\longrightarrow$ *i* $=$ *card* (*defer acc A p*) $-$ *1*
using *sequential-composition.simps snd-conv i-is-new-card*
by *metis*
hence *electoral-module acc* $\longrightarrow$ *i* $\geq$ *x*
using *card-too-big*
by *linarith*
hence *new-card-still-big-enough*: *electoral-module acc* $\longrightarrow$ *x* $\leq$ *i*
by *blast*
have
electoral-module acc $\wedge$ *electoral-module m* $\longrightarrow$
defer (*acc* $\triangleright$ *m*) *A p* $\subseteq$ *defer acc A p*
using *enough-leftover f-prof subset*
by *blast*
hence

```
  electoral-module acc ∧ electoral-module m ⟶
      i ≤ card (defer acc A p)
  using card-mono i-is-new-card step-profile
  by blast
hence i-geq-x:
  electoral-module acc ∧ electoral-module m ⟶ (i = x ∨ i > x)
  using nat-less-le new-card-still-big-enough
  by blast
thus ?thesis
proof cases
  assume new-card-greater-x: electoral-module acc ⟶ i > x
  hence electoral-module acc ⟶ 1 < card (defer (acc ▹ m) A p)
    using x-greater-zero i-is-new-card
    by linarith
  moreover have new-card-still-big-enough2:
    electoral-module acc ⟶ x ≤ i
    using i-is-new-card new-card-still-big-enough
    by blast
  moreover have
    n = card (defer acc A p) ⟶
        (electoral-module acc ⟶ i < n)
    using subset step-profile enough-leftover f-prof psubset-card-mono
          i-is-new-card
    by blast
  moreover have
    electoral-module acc ⟶
        electoral-module (acc ▹ m)
    using non-electing-def non-electing-m seq-comp-sound
    by blast
  moreover have non-electing-new:
    non-electing acc ⟶ non-electing (acc ▹ m)
    by (simp add: non-electing-m)
  ultimately have
    (n = card (defer acc A p) ∧ non-electing acc ∧
        electoral-module acc) ⟶
            card (defer (loop-comp-helper (acc ▹ m) m t) A p) = x
    using less.hyps i-is-new-card new-card-greater-x
    by blast
  thus ?thesis
    using f-prof rec-step non-electing-def
    by metis
next
  assume i-not-gt-x: ¬(electoral-module acc ⟶ i > x)
  hence electoral-module acc ∧ electoral-module m ⟶ i = x
    using i-geq-x
    by blast
  hence electoral-module acc ∧ electoral-module m ⟶ t ((acc ▹ m) A p)
    using i-is-new-card terminate-if-n-left
    by blast
```

hence
electoral-module acc $\wedge$ *electoral-module m* $\longrightarrow$
card (*defer-r* (*loop-comp-helper* (*acc* $\triangleright$ *m*) *m t A p*)) = *x*
using *loop-comp-helper.simps*(*1*) *terminate-if-n-left*
by *metis*
thus *?thesis*
using *i-not-gt-x dual-order.strict-iff-order i-is-new-card*
loop-comp-helper.simps(*1*) *new-card-still-big-enough*
f-prof rec-step terminate-if-n-left
by *metis*
qed
qed
qed
qed

lemma *loop-comp-helper-iter-elim-def-n*:
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall$ *r*. ((*t r*) $\longleftrightarrow$ (*card* (*defer-r r*) = *x*)) **and**
x-greater-zero: *x* > *0* **and**
f-prof: *finite-profile A p* **and**
acc-defers-enough: *card* (*defer acc A p*) $\geq$ *x* **and**
non-electing-acc: *non-electing acc*
shows *card* (*defer* (*loop-comp-helper acc m t*) *A p*) = *x*
using *acc-defers-enough gr-implies-not0 le-neq-implies-less*
less-one linorder-neqE-nat loop-comp-helper.simps(*1*)
loop-comp-helper-iter-elim-def-n-helper non-electing-acc
non-electing-m f-prof single-elimination nat-neq-iff
terminate-if-n-left x-greater-zero less-le
by (*metis* (*no-types*, *lifting*))

lemma *iter-elim-def-n-helper*:
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall$ *r*. ((*t r*) $\longleftrightarrow$ (*card* (*defer-r r*) = *x*)) **and**
x-greater-zero: *x* > *0* **and**
f-prof: *finite-profile A p* **and**
enough-alternatives: *card A* $\geq$ *x*
shows *card* (*defer* (*m* $\circlearrowleft_t$) *A p*) = *x*
proof *cases*
assume *card A* = *x*
thus *?thesis*
by (*simp add*: *terminate-if-n-left*)
next
assume *card-not-x*: $\neg$ *card A* = *x*
thus *?thesis*
proof *cases*

assume *card* $A < x$
thus *?thesis*
using *enough-alternatives not-le*
by *blast*
next
assume $\neg$*card* $A < x$
hence *card-big-enough-A*: *card* $A > x$
using *card-not-x*
by *linarith*
hence *card-m*: *card* (*defer* m A p) = *card* $A - 1$
using *non-electing-m f-prof single-elimination single-elim-decr-def-card2 x-greater-zero*
by *fastforce*
hence *card-big-enough-m*: *card* (*defer* m A p) $\geq x$
using *card-big-enough-A*
by *linarith*
hence (m $\circlearrowleft_t$) A p = (*loop-comp-helper* m m t) A p
by (*simp add*: *card-not-x terminate-if-n-left*)
thus *?thesis*
using *card-big-enough-m non-electing-m f-prof single-elimination terminate-if-n-left x-greater-zero loop-comp-helper-iter-elim-def-n*
by *metis*
qed
qed

4.5.4 Composition Rules

theorem *loop-comp-presv-def-lift-inv*[*simp*]:
assumes *monotone-m*: *defer-lift-invariance* m
shows *defer-lift-invariance* (m $\circlearrowleft_t$)
proof −
fix
A :: $'a$ *set*
have
$\forall$ p q a. ($a \in$ (*defer* (m $\circlearrowleft_t$) A p) $\wedge$ *lifted* A p q a) $\longrightarrow$ (m $\circlearrowleft_t$) A p = (m $\circlearrowleft_t$) A q
using *defer-module.simps monotone-m lifted-imp-fin-prof loop-composition.simps*(*1*) *loop-composition.simps*(*2*) *loop-comp-helper-def-lift-inv2*
by (*metis* (*full-types*))
thus *?thesis*
using *def-mod-def-lift-inv monotone-m loop-composition.simps*(*1*) *loop-composition.simps*(*2*) *defer-lift-invariance-def loop-comp-sound loop-comp-helper-def-lift-inv2 lifted-imp-fin-prof*
by (*smt* (*verit*, *best*))
qed

```
theorem loop-comp-presv-non-electing[simp]:
  assumes non-electing-m: non-electing m
  shows non-electing (m ↻t)
  unfolding non-electing-def
proof (safe, simp-all)
  show electoral-module (m ↻t)
    using loop-comp-sound non-electing-def non-electing-m
    by metis
next
    fix
      A :: 'a set and
      p :: 'a Profile and
      x :: 'a
    assume
      fin-A: finite A and
      prof-A: profile A p and
      x-elect: x ∈ elect (m ↻t) A p
    show False
  using def-mod-non-electing loop-comp-presv-non-electing-helper
        non-electing-m empty-iff fin-A loop-comp-code
        non-electing-def prof-A x-elect
  by metis
qed

theorem iter-elim-def-n[simp]:
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left: ∀ r. ((t r) ⟷ (card (defer-r r) = n)) and
    x-greater-zero: n > 0
  shows defers n (m ↻t)
proof −
  have
    ∀ A p. finite-profile A p ∧ card A ≥ n ⟶
        card (defer (m ↻t) A p) = n
    using iter-elim-def-n-helper non-electing-m single-elimination
          terminate-if-n-left x-greater-zero
    by blast
  moreover have electoral-module (m ↻t)
    using loop-comp-sound eliminates-def single-elimination
    by blast
  thus ?thesis
    by (simp add: calculation defers-def)
qed

end
```

4.6 Maximum Parallel Composition

theory *Maximum-Parallel-Composition*
  imports *Basic-Modules/Component-Types/Maximum-Aggregator*
    Parallel-Composition
begin

This is a family of parallel compositions. It composes a new electoral module from two electoral modules combined with the maximum aggregator. Therein, the two modules each make a decision and then a partition is returned where every alternative receives the maximum result of the two input partitions. This means that, if any alternative is elected by at least one of the modules, then it gets elected, if any non-elected alternative is deferred by at least one of the modules, then it gets deferred, only alternatives rejected by both modules get rejected.

4.6.1 Definition

fun *maximum-parallel-composition* :: *'a Electoral-Module* ⇒
    'a Electoral-Module ⇒ *'a Electoral-Module* **where**
  maximum-parallel-composition m n =
   (*let a* = *max-aggregator in* ($m \parallel_a n$))

abbreviation *max-parallel* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* ⇒
    'a Electoral-Module (**infix** $\parallel_\uparrow$ *50*) **where**
  $m \parallel_\uparrow n$ == *maximum-parallel-composition m n*

4.6.2 Soundness

theorem *max-par-comp-sound*:
  assumes
   mod-m: *electoral-module m* **and**
   mod-n: *electoral-module n*
  shows *electoral-module* ($m \parallel_\uparrow n$)
  using *mod-m mod-n*
  by *simp*

4.6.3 Lemmata

lemma *max-agg-eq-result*:
  assumes
   module-m: *electoral-module m* **and**
   module-n: *electoral-module n* **and**
   f-prof: *finite-profile A p* **and**
   in-A: $x \in A$
  shows

```
    mod-contains-result (m ∥↑ n) m A p x ∨
      mod-contains-result (m ∥↑ n) n A p x
proof cases
  assume a1: x ∈ elect (m ∥↑ n) A p
  hence
    let (e1, r1, d1) = m A p;
        (e2, r2, d2) = n A p in
      x ∈ e1 ∪ e2
    by auto
  hence x ∈ (elect m A p) ∪ (elect n A p)
    by auto
  thus ?thesis
    using IntI Un-iff a1 empty-iff mod-contains-result-def
          in-A max-agg-sound module-m module-n par-comp-sound
          f-prof result-disj maximum-parallel-composition.simps
    by (smt (verit, ccfv-threshold))
next
  assume not-a1: x ∉ elect (m ∥↑ n) A p
  thus ?thesis
  proof cases
    assume a2: x ∈ defer (m ∥↑ n) A p
    thus ?thesis
      using CollectD DiffD1 DiffD2 max-aggregator.simps Un-iff
            case-prod-conv defer-not-elec-or-rej max-agg-sound
            mod-contains-result-def module-m module-n par-comp-sound
            parallel-composition.simps prod.collapse f-prof sndI
            Int-iff electoral-mod-defer-elem electoral-module-def
            max-agg-rej-set prod.sel(1) maximum-parallel-composition.simps
      by (smt (verit, del-insts))
  next
    assume not-a2: x ∉ defer (m ∥↑ n) A p
    with not-a1 have a3:
      x ∈ reject (m ∥↑ n) A p
      using electoral-mod-defer-elem in-A max-agg-sound module-m module-n
            par-comp-sound f-prof maximum-parallel-composition.simps
      by metis
    hence
      let (e1, r1, d1) = m A p;
          (e2, r2, d2) = n A p in
        x ∈ fst (snd (max-aggregator A (e1, r1, d1) (e2, r2, d2)))
      using case-prod-unfold parallel-composition.simps
            surjective-pairing maximum-parallel-composition.simps
      by (smt (verit, ccfv-threshold))
    hence
      let (e1, r1, d1) = m A p;
          (e2, r2, d2) = n A p in
        x ∈ A − (e1 ∪ e2 ∪ d1 ∪ d2)
      by simp
    thus ?thesis
```

using *Un-iff combine-ele-rej-def agg-conservative-def contra-subsetD disjoint-iff-not-equal in-A electoral-module-def mod-contains-result-def max-agg-consv module-m module-n par-comp-sound parallel-composition.simps f-prof result-disj max-agg-rej-set not-a1 not-a2 Int-iff maximum-parallel-composition.simps*
by (*smt* (*verit*, *del-insts*))
qed
qed

lemma *max-agg-rej-iff-both-reject*:
assumes
f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n*
shows
$x \in reject\ (m \parallel_{\uparrow} n)\ A\ p \longleftrightarrow$
$(x \in reject\ m\ A\ p \wedge x \in reject\ n\ A\ p)$
proof −
have
$x \in reject\ (m \parallel_{\uparrow} n)\ A\ p \longrightarrow$
$(x \in reject\ m\ A\ p \wedge x \in reject\ n\ A\ p)$
proof
assume *a*: $x \in reject\ (m \parallel_{\uparrow} n)\ A\ p$
hence
let $(e1, r1, d1) = m\ A\ p;$
$(e2, r2, d2) = n\ A\ p$ *in*
$x \in fst\ (snd\ (max\text{-}aggregator\ A\ (e1, r1, d1)\ (e2, r2, d2)))$
using *case-prodI2 maximum-parallel-composition.simps split-def parallel-composition.simps prod.collapse split-beta*
by (*smt* (*verit*, *ccfv-threshold*))
hence
let $(e1, r1, d1) = m\ A\ p;$
$(e2, r2, d2) = n\ A\ p$ *in*
$x \in A - (e1 \cup e2 \cup d1 \cup d2)$
by *simp*
thus $x \in reject\ m\ A\ p \wedge x \in reject\ n\ A\ p$
using *Int-iff a electoral-module-def max-agg-rej-set module-m module-n parallel-composition.simps surjective-pairing maximum-parallel-composition.simps f-prof*
by (*smt* (*verit*, *best*))
qed
moreover have
$(x \in reject\ m\ A\ p \wedge x \in reject\ n\ A\ p) \longrightarrow$
$x \in reject\ (m \parallel_{\uparrow} n)\ A\ p$
proof
assume *a*: $x \in reject\ m\ A\ p \wedge x \in reject\ n\ A\ p$
hence

$x \notin$ *elect m A p* $\wedge$ $x \notin$ *defer m A p* $\wedge$
$x \notin$ *elect n A p* $\wedge$ $x \notin$ *defer n A p*
using *IntI empty-iff module-m module-n f-prof result-disj*
by *metis*
thus $x \in$ *reject* $(m \parallel_\uparrow n)$ *A p*
using *CollectD DiffD1 max-aggregator.simps Un-iff a electoral-mod-defer-elem prod.simps max-agg-sound module-m module-n f-prof old.prod.inject par-comp-sound prod.collapse parallel-composition.simps reject-not-elec-or-def maximum-parallel-composition.simps*
by (*smt* (*verit*, *ccfv-threshold*))
qed
ultimately show *?thesis*
by *blast*
qed

lemma *max-agg-rej1*:
assumes
f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
rejected: $x \in$ *reject n A p*
shows
mod-contains-result m $(m \parallel_\uparrow n)$ *A p x*
using *Set.set-insert contra-subsetD disjoint-insert mod-contains-result-comm mod-contains-result-def max-agg-eq-result max-agg-rej-iff-both-reject module-m module-n f-prof reject-in-alts rejected result-disj*
by (*smt* (*verit*, *best*))

lemma *max-agg-rej2*:
assumes
f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
rejected: $x \in$ *reject n A p*
shows
mod-contains-result $(m \parallel_\uparrow n)$ *m A p x*
using *mod-contains-result-comm max-agg-rej1 module-m module-n f-prof rejected*
by *metis*

lemma *max-agg-rej3*:
assumes
f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
rejected: $x \in$ *reject m A p*

shows
mod-contains-result n $(m \parallel_\uparrow n)$ *A p x*
using *contra-subsetD disjoint-iff-not-equal result-disj*
mod-contains-result-comm mod-contains-result-def
max-agg-eq-result max-agg-rej-iff-both-reject
module-m module-n f-prof reject-in-alts rejected
by (*smt* (*verit*, *ccfv-SIG*))

lemma *max-agg-rej4*:
assumes
f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
rejected: $x \in$ *reject m A p*
shows
mod-contains-result $(m \parallel_\uparrow n)$ *n A p x*
using *mod-contains-result-comm max-agg-rej3*
module-m module-n f-prof rejected
by *metis*

lemma *max-agg-rej-intersect*:
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows
reject $(m \parallel_\uparrow n)$ *A p* =
(*reject m A p*) ∩ (*reject n A p*)
proof −
have
A = (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) ∧
A = (*elect n A p*) ∪ (*reject n A p*) ∪ (*defer n A p*)
by (*simp add*: *module-m module-n f-prof result-presv-alts*)
hence
A − ((*elect m A p*) ∪ (*defer m A p*)) = (*reject m A p*) ∧
A − ((*elect n A p*) ∪ (*defer n A p*)) = (*reject n A p*)
using *module-m module-n f-prof reject-not-elec-or-def*
by *auto*
hence
A − ((*elect m A p*) ∪ (*elect n A p*) ∪ (*defer m A p*) ∪ (*defer n A p*)) =
(*reject m A p*) ∩ (*reject n A p*)
by *blast*
hence
let (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
A − (*e1* ∪ *e2* ∪ *d1* ∪ *d2*) = *r1* ∩ *r2*
by *fastforce*
thus *?thesis*
by *auto*

qed

lemma *dcompat-dec-by-one-mod*:
  assumes
    compatible: *disjoint-compatibility* m n **and**
    in-A: $x \in A$
  shows
    $(\forall\, p.$ *finite-profile* A $p \longrightarrow$
        mod-contains-result m $(m \parallel_\uparrow n)$ A p $x) \vee$
      $(\forall\, p.$ *finite-profile* A $p \longrightarrow$
        mod-contains-result n $(m \parallel_\uparrow n)$ A p $x)$
  using *DiffI compatible disjoint-compatibility-def*
      in-A max-agg-rej1 max-agg-rej3
  by *metis*

4.6.4 Composition Rules

theorem *conserv-max-agg-presv-non-electing*[*simp*]:
  assumes
    non-electing-m: *non-electing* m **and**
    non-electing-n: *non-electing* n
  shows *non-electing* $(m \parallel_\uparrow n)$
  using *non-electing-m non-electing-n*
  by *simp*

theorem *par-comp-def-lift-inv*[*simp*]:
  assumes
    compatible: *disjoint-compatibility* m n **and**
    monotone-m: *defer-lift-invariance* m **and**
    monotone-n: *defer-lift-invariance* n
  shows *defer-lift-invariance* $(m \parallel_\uparrow n)$
  unfolding *defer-lift-invariance-def*
proof (*safe*)
  have *electoral-mod-m*: *electoral-module* m
    using *monotone-m*
    by (*simp add*: *defer-lift-invariance-def*)
  have *electoral-mod-n*: *electoral-module* n
    using *monotone-n*
    by (*simp add*: *defer-lift-invariance-def*)
  show *electoral-module* $(m \parallel_\uparrow n)$
    using *electoral-mod-m electoral-mod-n*
    by *simp*
next
  fix
    S :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    q :: $'a$ *Profile* **and**
    x :: $'a$

assume
defer-x: $x \in$ *defer* $(m \parallel_{\uparrow} n)$ S p **and**
lifted-x: *Profile.lifted* S p q x
hence *f-profs*: *finite-profile* S p $\wedge$ *finite-profile* S q
by (*simp add*: *lifted-def*)
from *compatible* **obtain** A::$'a$ *set* **where** A:
$A \subseteq S \wedge (\forall x \in A.$ *indep-of-alt* m S $x \wedge$
$(\forall p.$ *finite-profile* S $p \longrightarrow x \in$ *reject* m S $p)) \wedge$
$(\forall x \in S{-}A.$ *indep-of-alt* n S $x \wedge$
$(\forall p.$ *finite-profile* S $p \longrightarrow x \in$ *reject* n S $p))$
using *disjoint-compatibility-def f-profs*
by (*metis* (*no-types*, *lifting*))
have
$\forall x \in S.$ *prof-contains-result* $(m \parallel_{\uparrow} n)$ S p q x
proof *cases*
assume *a0*: $x \in A$
hence $x \in$ *reject* m S p
using A *f-profs*
by *blast*
with *defer-x* **have** *defer-n*: $x \in$ *defer* n S p
using *compatible disjoint-compatibility-def*
mod-contains-result-def f-profs max-agg-rej4
by *metis*
have
$\forall x \in A.$ *mod-contains-result* $(m \parallel_{\uparrow} n)$ n S p x
using A *compatible disjoint-compatibility-def*
max-agg-rej4 f-profs
by *metis*
moreover have $\forall x \in S.$ *prof-contains-result* n S p q x
using *defer-n lifted-x prof-contains-result-def monotone-n f-profs*
defer-lift-invariance-def
by (*smt* (*verit*, *del-insts*))
moreover have
$\forall x \in A.$ *mod-contains-result* n $(m \parallel_{\uparrow} n)$ S q x
using A *compatible disjoint-compatibility-def*
max-agg-rej3 f-profs
by *metis*
ultimately have *00*:
$\forall x \in A.$ *prof-contains-result* $(m \parallel_{\uparrow} n)$ S p q x
by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
have
$\forall x \in S{-}A.$ *mod-contains-result* $(m \parallel_{\uparrow} n)$ m S p x
using A *max-agg-rej2 monotone-m monotone-n f-profs*
defer-lift-invariance-def
by *metis*
moreover have $\forall x \in S.$ *prof-contains-result* m S p q x
using A *lifted-x a0 prof-contains-result-def indep-of-alt-def*
lifted-imp-equiv-prof-except-a f-profs IntI
electoral-mod-defer-elem empty-iff result-disj

by (*smt* (*verit*, *ccfv-threshold*))
moreover have
$\forall x \in S{-}A$. *mod-contains-result* m (m $\|_\uparrow$ n) S q x
using *A max-agg-rej1 monotone-m monotone-n f-profs*
defer-lift-invariance-def
by *metis*
ultimately have *01*:
$\forall x \in S{-}A$. *prof-contains-result* (m $\|_\uparrow$ n) S p q x
by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
from *00 01*
show *?thesis*
by *blast*
next
assume $x \notin A$
hence *a1*: $x \in S{-}A$
using *DiffI lifted-x compatible f-profs*
Profile.lifted-def
by (*metis* (*no-types*, *lifting*))
hence $x \in$ *reject* n S p
using *A f-profs*
by *blast*
with *defer-x* **have** *defer-n*: $x \in$ *defer* m S p
using *DiffD1 DiffD2 compatible dcompat-dec-by-one-mod*
defer-not-elec-or-rej disjoint-compatibility-def
not-rej-imp-elec-or-def mod-contains-result-def
max-agg-sound par-comp-sound f-profs
maximum-parallel-composition.simps
by *metis*
have
$\forall x \in A$. *mod-contains-result* (m $\|_\uparrow$ n) n S p x
using *A compatible disjoint-compatibility-def*
max-agg-rej4 f-profs
by *metis*
moreover have $\forall x \in S$. *prof-contains-result* n S p q x
using *A lifted-x a1 prof-contains-result-def indep-of-alt-def*
lifted-imp-equiv-prof-except-a f-profs
electoral-mod-defer-elem
by (*smt* (*verit*, *ccfv-threshold*))
moreover have
$\forall x \in A$. *mod-contains-result* n (m $\|_\uparrow$ n) S q x
using *A compatible disjoint-compatibility-def*
max-agg-rej3 f-profs
by *metis*
ultimately have *10*:
$\forall x \in A$. *prof-contains-result* (m $\|_\uparrow$ n) S p q x
by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
have
$\forall x \in S{-}A$. *mod-contains-result* (m $\|_\uparrow$ n) m S p x
using *A max-agg-rej2 monotone-m monotone-n*

f-profs defer-lift-invariance-def
by *metis*
moreover have $\forall x \in S.$ *prof-contains-result m S p q x*
using *lifted-x defer-n prof-contains-result-def monotone-m*
f-profs defer-lift-invariance-def
by (*smt* (*verit*, *ccfv-threshold*))
moreover have
$\forall x \in S{-}A.$ *mod-contains-result m* ($m \parallel_\uparrow n$) *S q x*
using *A max-agg-rej1 monotone-m monotone-n*
f-profs defer-lift-invariance-def
by *metis*
ultimately have *11*:
$\forall x \in S{-}A.$ *prof-contains-result* ($m \parallel_\uparrow n$) *S p q x*
using *electoral-mod-defer-elem*
by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
from *10 11*
show *?thesis*
by *blast*
qed
thus ($m \parallel_\uparrow n$) $S\ p$ = ($m \parallel_\uparrow n$) $S\ q$
using *compatible disjoint-compatibility-def f-profs*
eq-alts-in-profs-imp-eq-results max-par-comp-sound
by *metis*
qed

lemma *par-comp-rej-card*:
assumes
compatible: *disjoint-compatibility x y* **and**
f-prof: *finite-profile S p* **and**
reject-sum: *card* (*reject x S p*) + *card* (*reject y S p*) = *card S* + *n*
shows *card* (*reject* ($x \parallel_\uparrow y$) *S p*) = *n*
proof −
from *compatible* **obtain** *A* **where** *A*:
$A \subseteq S \wedge$
($\forall a \in A.$ *indep-of-alt x S a* $\wedge$
($\forall p.$ *finite-profile S p* $\longrightarrow a \in$ *reject x S p*)) $\wedge$
($\forall a \in S{-}A.$ *indep-of-alt y S a* $\wedge$
($\forall p.$ *finite-profile S p* $\longrightarrow a \in$ *reject y S p*))
using *disjoint-compatibility-def f-prof*
by *metis*
from *f-prof compatible*
have *reject-representation*:
reject ($x \parallel_\uparrow y$) *S p* = (*reject x S p*) $\cap$ (*reject y S p*)
using *max-agg-rej-intersect disjoint-compatibility-def*
by *blast*
have *electoral-module x* $\wedge$ *electoral-module y*
using *compatible disjoint-compatibility-def*
by *auto*
hence *subsets*: (*reject x S p*) $\subseteq S \wedge$ (*reject y S p*) $\subseteq S$

```
    by (simp add: f-prof reject-in-alts)
  hence finite (reject x S p) ∧ finite (reject y S p)
    using rev-finite-subset f-prof reject-in-alts
    by auto
  hence 0:
    card (reject (x ∥↑ y) S p) =
        card S + n −
          card ((reject x S p) ∪ (reject y S p))
    using card-Un-Int reject-representation reject-sum
    by fastforce
  have ∀ a ∈ S. a ∈ (reject x S p) ∨ a ∈ (reject y S p)
    using A f-prof
    by blast
  hence 1: card ((reject x S p) ∪ (reject y S p)) = card S
    using subsets subset-eq sup.absorb-iff1
        sup.cobounded1 sup-left-commute
    by (smt (verit, best))
  from 0 1
  show card (reject (x ∥↑ y) S p) = n
    by simp
qed


theorem par-comp-elim-one[simp]:
  assumes
    defers-m-1: defers 1 m and
    non-elec-m: non-electing m and
    rejec-n-2: rejects 2 n and
    disj-comp: disjoint-compatibility m n
  shows eliminates 1 (m ∥↑ n)
  unfolding eliminates-def
proof (safe)
  have electoral-mod-m: electoral-module m
    using non-elec-m
    by (simp add: non-electing-def)
  have electoral-mod-n: electoral-module n
    using rejec-n-2
    by (simp add: rejects-def)
  show electoral-module (m ∥↑ n)
    using electoral-mod-m electoral-mod-n
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile
  assume
    min-2-card: 1 < card A and
    fin-A: finite A and
    prof-A: profile A p
```

```
have card-geq-1: card A ≥ 1
  using min-2-card dual-order.strict-trans2 less-imp-le-nat
  by blast
have module: electoral-module m
  using non-elec-m non-electing-def
  by auto
have elec-card-0: card (elect m A p) = 0
  using fin-A prof-A non-elec-m card-eq-0-iff non-electing-def
  by metis
moreover
from card-geq-1 have def-card-1:
  card (defer m A p) = 1
  using defers-m-1 module fin-A prof-A
  by (simp add: defers-def)
ultimately have card-reject-m:
  card (reject m A p) = card A − 1
proof −
  have finite A
    by (simp add: fin-A)
  moreover have
    well-formed A
      (elect m A p, reject m A p, defer m A p)
    using fin-A prof-A electoral-module-def module
    by auto
  ultimately have
    card A =
      card (elect m A p) + card (reject m A p) +
        card (defer m A p)
    using result-count
    by blast
  thus ?thesis
    using def-card-1 elec-card-0
    by simp
qed
have case1: card A ≥ 2
  using min-2-card
  by auto
from case1 have card-reject-n:
  card (reject n A p) = 2
  using fin-A prof-A rejec-n-2 rejects-def
  by blast
from card-reject-m card-reject-n
have
  card (reject m A p) + card (reject n A p) =
    card A + 1
  using card-geq-1
  by linarith
with disj-comp prof-A fin-A card-reject-m card-reject-n
show
```

card (*reject* (*m* $\|_\uparrow$ *n*) *A p*) = *1*
using *par-comp-rej-card*
by *blast*
qed

end

4.7 Elect Composition

theory *Elect-Composition*
imports *Basic-Modules/Elect-Module*
Sequential-Composition
begin

The elect composition sequences an electoral module and the elect module. It finalizes the module's decision as it simply elects all their non-rejected alternatives. Thereby, any such elect-composed module induces a proper voting rule in the social choice sense, as all alternatives are either rejected or elected.

4.7.1 Definition

fun *elector* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* **where**
elector m = (*m* ▷ *elect-module*)

4.7.2 Soundness

theorem *elector-sound*[*simp*]:
assumes *module-m*: *electoral-module m*
shows *electoral-module* (*elector m*)
by (*simp add*: *module-m*)

4.7.3 Electing

theorem *elector-electing*[*simp*]:
assumes
module-m: *electoral-module m* **and**
non-block-m: *non-blocking m*
shows *electing* (*elector m*)
proof −
obtain
AA :: *'a Electoral-Module* ⇒ *'a set* **and**
rrs :: *'a Electoral-Module* ⇒ *'a Profile* **where**
f1:

$\forall f.$
  $(electing\ f \lor$
    $\{\} = elect\ f\ (AA\ f)\ (rrs\ f) \land profile\ (AA\ f)\ (rrs\ f) \land$
      $finite\ (AA\ f) \land \{\} \neq AA\ f \lor$
    $\neg\ electoral\text{-}module\ f) \land$
      $((\forall A\ rs.\ \{\} \neq elect\ f\ A\ rs \lor \neg\ profile\ A\ rs \lor$
        $infinite\ A \lor \{\} = A) \land$
      $electoral\text{-}module\ f \lor$
    $\neg\ electing\ f)$
using *electing-def*
by *metis*
have *non-block*:
non-blocking
  $(elect\text{-}module::'a\ set \Rightarrow \text{-}\ Profile \Rightarrow \text{-}\ Result)$
by (*simp add*: *electing-imp-non-blocking*)
thus *?thesis*

proof −
obtain
  $AAa :: 'a\ Electoral\text{-}Module \Rightarrow 'a\ set$ **and**
  $rrsa :: 'a\ Electoral\text{-}Module \Rightarrow 'a\ Profile$ **where**
  f1:
  $\forall f.$
    $(electing\ f \lor$
      $\{\} = elect\ f\ (AAa\ f)\ (rrsa\ f) \land profile\ (AAa\ f)\ (rrsa\ f) \land$
        $finite\ (AAa\ f) \land \{\} \neq AAa\ f \lor$
    $\neg\ electoral\text{-}module\ f) \land ((\forall A\ rs.\ \{\} \neq elect\ f\ A\ rs \lor$
    $\neg\ profile\ A\ rs \lor infinite\ A \lor \{\} = A) \land electoral\text{-}module\ f \lor$
    $\neg\ electing\ f)$
  using *electing-def*
  by *metis*
obtain
  $AAb :: 'a\ Result \Rightarrow 'a\ set$ **and**
  $AAc :: 'a\ Result \Rightarrow 'a\ set$ **and**
  $AAd :: 'a\ Result \Rightarrow 'a\ set$ **where**
  f2:
  $\forall p.\ (AAb\ p,\ AAc\ p,\ AAd\ p) = p$
  using *disjoint3.cases*
  by (*metis* (*no-types*))
have *f3*:
  $electoral\text{-}module\ (elector\ m)$
  using *elector-sound module-m*
  by *simp*
have *f4*:
  $\forall p.\ (elect\text{-}r\ p,\ AAc\ p,\ AAd\ p) = p$
  using *f2*
  by *simp*
have
  $finite\ (AAa\ (elector\ m)) \land$

profile $(AAa\ (elector\ m))\ (rrsa\ (elector\ m)) \wedge$
$\{\} = elect\ (elector\ m)\ (AAa\ (elector\ m))\ (rrsa\ (elector\ m)) \wedge$
$\{\} = AAd\ (elector\ m\ (AAa\ (elector\ m))\ (rrsa\ (elector\ m))) \wedge$
reject $(elector\ m)\ (AAa\ (elector\ m))\ (rrsa\ (elector\ m)) =$
$AAc\ (elector\ m\ (AAa\ (elector\ m))\ (rrsa\ (elector\ m))) \longrightarrow$
electing $(elector\ m)$
using *f2 f1 Diff-empty elector.simps non-block-m snd-conv non-blocking-def reject-not-elec-or-def non-block seq-comp-presv-non-blocking*
by *metis*
moreover
{
assume
$\{\} \neq AAd\ (elector\ m\ (AAa\ (elector\ m))\ (rrsa\ (elector\ m)))$
hence
$\neg$ *profile* $(AAa\ (elector\ m))\ (rrsa\ (elector\ m)) \vee$
infinite $(AAa\ (elector\ m))$
using *f4*
by *simp*
}
ultimately show *?thesis*
using *f4 f3 f1 fst-conv snd-conv*
by *metis*
qed
qed

4.7.4 Composition Rule

lemma *dcc-imp-cc-elector*:
assumes *dcc*: *defer-condorcet-consistency m*
shows *condorcet-consistency* (*elector m*)
proof (*unfold defer-condorcet-consistency-def condorcet-consistency-def*, *auto*)
show *electoral-module* ($m \triangleright$ *elect-module*)
using *dcc defer-condorcet-consistency-def elect-mod-sound seq-comp-sound*
by *metis*
next
show
$\bigwedge A\ p\ w\ x.$
finite $A \Longrightarrow$ *profile* $A\ p \Longrightarrow w \in A \Longrightarrow$
$\forall x{\in}A - \{w\}.\ card\ \{i.\ i < length\ p \wedge (w, x) \in (p!i)\} <$
$card\ \{i.\ i < length\ p \wedge (x, w) \in (p!i)\} \Longrightarrow$
$x \in elect\ m\ A\ p \Longrightarrow x \in A$
proof −
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a* **and**

```
      x :: 'a
    assume
      finite: finite A and
      prof-A: profile A p
    show
      ∀ y∈A − {w}.
          card {i. i < length p ∧ (w, y) ∈ (p!i)} <
            card {i. i < length p ∧ (y, w) ∈ (p!i)} ⟹
             x ∈ elect m A p ⟹ x ∈ A
      using dcc defer-condorcet-consistency-def
            elect-in-alts subset-eq finite prof-A
      by metis
  qed
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a and
    x :: 'a and
    xa :: 'a
  assume
    finite: finite A and
    prof-A: profile A p and
    w-in-A: w ∈ A and
    1: x ∈ elect m A p and
    2: ∀ y∈A − {w}.
          card {i. i < length p ∧ (w, y) ∈ (p!i)} <
            card {i. i < length p ∧ (y, w) ∈ (p!i)}
  have condorcet-winner A p w
    using finite prof-A w-in-A 2
    by simp
  thus xa = x
    using condorcet-winner.simps dcc fst-conv insert-Diff 1
          defer-condorcet-consistency-def insert-not-empty
    by (metis (no-types, lifting))
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a and
    x :: 'a
  assume
    finite: finite A and
    prof-A: profile A p and
    w-in-A: w ∈ A and
    0: ∀ y∈A − {w}.
          card {i. i < length p ∧ (w, y) ∈ (p!i)} <
            card {i. i < length p ∧ (y, w) ∈ (p!i)} and
    1: x ∈ defer m A p
```

have *condorcet-winner A p w*
using *finite prof-A w-in-A 0*
by *simp*
thus $x \in A$
using *0 1 condorcet-winner.simps dcc defer-in-alts defer-condorcet-consistency-def order-trans subset-Compl-singleton*
by (*metis* (*no-types*, *lifting*))
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$w :: 'a$ **and**
$x :: 'a$ **and**
$xa :: 'a$
assume
finite: *finite A* **and**
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
1: $x \in defer\ m\ A\ p$ **and**
xa-in-A: $xa \in A$ **and**
2: $\forall y{\in}A - \{w\}.$
$card\ \{i.\ i < length\ p \wedge (w, y) \in (p!i)\} <$
$card\ \{i.\ i < length\ p \wedge (y, w) \in (p!i)\}$ **and**
3: $\neg\ card\ \{i.\ i < length\ p \wedge (x, xa) \in (p!i)\} <$
$card\ \{i.\ i < length\ p \wedge (xa, x) \in (p!i)\}$
have *condorcet-winner A p w*
using *finite prof-A w-in-A 2*
by *simp*
thus $xa = x$
using *1 2 condorcet-winner.simps dcc empty-iff xa-in-A defer-condorcet-consistency-def 3 DiffI cond-winner-unique3 insert-iff prod.sel*(*2*)
by (*metis* (*no-types*, *lifting*))
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$w :: 'a$ **and**
$x :: 'a$
assume
finite: *finite A* **and**
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
x-in-A: $x \in A$ **and**
1: $x \notin defer\ m\ A\ p$ **and**
2: $\forall y{\in}A - \{w\}.$
$card\ \{i.\ i < length\ p \wedge (w, y) \in (p!i)\} <$
$card\ \{i.\ i < length\ p \wedge (y, w) \in (p!i)\}$ **and**

3: $\forall\, y \in A - \{x\}$.
      $card\ \{i.\ i < length\ p \wedge (x,\ y) \in (p!i)\} <$
       $card\ \{i.\ i < length\ p \wedge (y,\ x) \in (p!i)\}$
have *condorcet-winner A p w*
  using *finite prof-A w-in-A 2*
  by *simp*
also have *condorcet-winner A p x*
  using *finite prof-A x-in-A 3*
  by *simp*
ultimately show $x \in elect\ m\ A\ p$
  using *1 condorcet-winner.simps dcc*
      defer-condorcet-consistency-def
      cond-winner-unique3 insert-iff eq-snd-iff
  by (*metis* (*no-types*, *lifting*))
next
fix
  $A :: 'a\ set$ **and**
  $p :: 'a\ Profile$ **and**
  $w :: 'a$ **and**
  $x :: 'a$
assume
  finite: *finite A* **and**
  prof-A: *profile A p* **and**
  w-in-A: $w \in A$ **and**
  1: $x \in reject\ m\ A\ p$ **and**
  2: $\forall\, y \in A - \{w\}$.
      $card\ \{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$
       $card\ \{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\}$
have *condorcet-winner A p w*
  using *finite prof-A w-in-A 2*
  by *simp*
thus $x \in A$
  using *1 dcc defer-condorcet-consistency-def finite*
      prof-A reject-in-alts subsetD
  by *metis*
next
fix
  $A :: 'a\ set$ **and**
  $p :: 'a\ Profile$ **and**
  $w :: 'a$ **and**
  $x :: 'a$
assume
  finite: *finite A* **and**
  prof-A: *profile A p* **and**
  w-in-A: $w \in A$ **and**
  0: $x \in reject\ m\ A\ p$ **and**
  1: $x \in elect\ m\ A\ p$ **and**
  2: $\forall\, y \in A - \{w\}$.
      $card\ \{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$

card $\{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\}$
have *condorcet-winner A p w*
using *finite prof-A w-in-A 2*
by *simp*
thus *False*
using *0 1 condorcet-winner.simps dcc IntI empty-iff*
defer-condorcet-consistency-def result-disj
by (*metis* (*no-types*, *hide-lams*))
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a* **and**
x :: *'a*
assume
finite: *finite A* **and**
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
0: $x \in$ *reject m A p* **and**
1: $x \in$ *defer m A p* **and**
2: $\forall\, y{\in}A - \{w\}.$
card $\{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$
card $\{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\}$
have *condorcet-winner A p w*
using *finite prof-A w-in-A 2*
by *simp*
thus *False*
using *0 1 dcc defer-condorcet-consistency-def IntI*
Diff-empty Diff-iff finite prof-A result-disj
by (*metis* (*no-types*, *hide-lams*))
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a* **and**
x :: *'a*
assume
finite: *finite A* **and**
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
x-in-A: $x \in A$ **and**
0: $x \notin$ *reject m A p* **and**
1: $x \notin$ *defer m A p* **and**
2: $\forall\, y{\in}A - \{w\}.$
card $\{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$
card $\{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\}$
have *condorcet-winner A p w*
using *finite prof-A w-in-A 2*
by *simp*

thus $x \in$ *elect m A p*
  using *0 1 condorcet-winner.simps dcc x-in-A*
    defer-condorcet-consistency-def electoral-mod-defer-elem
  by (*metis* (*no-types*, *lifting*))
qed

end

4.8 Defer One Loop Composition

theory *Defer-One-Loop-Composition*
  imports *Basic-Modules/Component-Types/Defer-Equal-Condition*
    Loop-Composition
    Elect-Composition
begin

This is a family of loop compositions. It uses the same module in sequence until either no new decisions are made or only one alternative is remaining in the defer-set. The second family herein uses the above family and subsequently elects the remaining alternative.

4.8.1 Definition

fun *iter* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* **where**
  iter m =
    (*let t* = *defer-equal-condition 1 in*
      (*m* $\circlearrowleft_t$))

abbreviation *defer-one-loop* ::
  'a Electoral-Module $\Rightarrow$ *'a Electoral-Module*
    (-$\circlearrowleft_{\exists !d}$ *50*) **where**
  m $\circlearrowleft_{\exists !d}$ $\equiv$ *iter m*

fun *iterelect* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* **where**
  iterelect m = *elector* (*m* $\circlearrowleft_{\exists !d}$)

end

Chapter 5

Voting Rules

5.1 Borda Rule

theory *Borda-Rule*
  imports *Compositional-Structures/Basic-Modules/Borda-Module*
    Compositional-Structures/Elect-Composition
begin

This is the Borda rule. On each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected.

5.1.1 Definition

fun *borda-rule* :: *'a Electoral-Module* **where**
  borda-rule A p = elector borda A p

fun *borda-rule-code* :: *'a Electoral-Module* **where**
  borda-rule-code A p = elector borda-code A p

end

5.2 Pairwise Majority Rule

theory *Pairwise-Majority-Rule*
  imports *Compositional-Structures/Basic-Modules/Condorcet-Module*
    Compositional-Structures/Defer-One-Loop-Composition
begin

This is the pairwise majority rule, a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives.

5.2.1 Definition

fun *pairwise-majority-rule* :: *'a Electoral-Module* **where**
pairwise-majority-rule A p = elector condorcet A p

fun *pairwise-majority-rule-code* :: *'a Electoral-Module* **where**
pairwise-majority-rule-code A p = elector condorcet-code A p

fun *condorcet'* :: *'a Electoral-Module* **where**
condorcet' A p =
((*min-eliminator condorcet-score*) $\circlearrowleft_{\exists !d}$) *A p*

fun *pairwise-majority-rule'* :: *'a Electoral-Module* **where**
pairwise-majority-rule' A p = iterelect condorcet' A p

5.2.2 Condorcet Consistency Property

theorem *condorcet-condorcet*: *condorcet-consistency pairwise-majority-rule*
proof −
have
condorcet-consistency (*elector condorcet*)
using *condorcet-is-dcc dcc-imp-cc-elector*
by *metis*
thus *?thesis*
using *condorcet-consistency2 electoral-module-def*
pairwise-majority-rule.simps
by *metis*
qed

end

5.3 Copeland Rule

theory *Copeland-Rule*
imports *Compositional-Structures/Basic-Modules/Copeland-Module*
Compositional-Structures/Elect-Composition
begin

This is the Copeland voting rule. The idea is to elect the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses.

5.3.1 Definition

fun *copeland-rule* :: *'a Electoral-Module* **where**
copeland-rule A p = elector copeland A p

fun *copeland-rule-code* :: *'a Electoral-Module* **where**
copeland-rule-code A p = elector copeland-code A p

5.3.2 Condorcet Consistency Property

theorem *copeland-condorcet*: *condorcet-consistency copeland-rule*
proof –
have
condorcet-consistency (*elector copeland*)
using *copeland-is-dcc dcc-imp-cc-elector*
by *metis*
thus *?thesis*
using *condorcet-consistency2 electoral-module-def copeland-rule.simps*
by *metis*
qed

end

5.4 Minimax Rule

theory *Minimax-Rule*
imports *Compositional-Structures/Basic-Modules/Minimax-Module Compositional-Structures/Elect-Composition*
begin

This is the Minimax voting rule. It elects the alternatives with the highest Minimax score.

5.4.1 Definition

fun *minimax-rule* :: *'a Electoral-Module* **where**
minimax-rule A p = elector minimax A p

fun *minimax-rule-code* :: *'a Electoral-Module* **where**
minimax-rule-code A p = elector minimax-code A p

5.4.2 Condorcet Consistency Property

theorem *minimax-condorcet*: *condorcet-consistency minimax-rule*
proof –

have
condorcet-consistency (*elector minimax*)
using *minimax-is-dcc dcc-imp-cc-elector*
by *metis*
thus *?thesis*
using *condorcet-consistency2 electoral-module-def minimax-rule.simps*
by *metis*
qed

end

5.5 Black's Rule

theory *Blacks-Rule*
imports *Pairwise-Majority-Rule*
Borda-Rule
begin

This is Black's voting rule. It is composed of a function that determines the Condorcet winner, i.e., the Pairwise Majority rule, and the Borda rule. Whenever there exists no Condorcet winner, it elects the choice made by the Borda rule, otherwise the Condorcet winner is elected.

5.5.1 Definition

fun *blacks-rule* :: *'a Electoral-Module* **where**
blacks-rule A p = (*pairwise-majority-rule* ▷ *borda-rule*) *A p*

fun *blacks-rule-code* :: *'a Electoral-Module* **where**
blacks-rule-code A p = (*pairwise-majority-rule-code* ▷ *borda-rule-code*) *A p*

end

5.6 Nanson-Baldwin Rule

theory *Nanson-Baldwin-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
Compositional-Structures/Defer-One-Loop-Composition
begin

This is the Nanson-Baldwin voting rule. It excludes alternatives with the lowest Borda score from the set of possible winners and then adjusts the Borda score to the new (remaining) set of still eligible alternatives.

5.6.1 Definition

fun *nanson-baldwin-rule* :: *'a Electoral-Module* **where**
  nanson-baldwin-rule A p =
    ((*min-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) *A p*

fun *nanson-baldwin-rule-code* :: *'a Electoral-Module* **where**
  nanson-baldwin-rule-code A p =
    ((*min-eliminator borda-score-code*) $\circlearrowleft_{\exists!d}$) *A p*

end

5.7 Classic Nanson Rule

theory *Classic-Nanson-Rule*
  imports *Compositional-Structures/Basic-Modules/Borda-Module*
    Compositional-Structures/Defer-One-Loop-Composition
begin

This is the classic Nanson's voting rule, i.e., the rule that was originally invented by Nanson, but not the Nanson-Baldwin rule. The idea is similar, however, as alternatives with a Borda score less or equal than the average Borda score are excluded. The Borda scores of the remaining alternatives are hence adjusted to the new set of (still) eligible alternatives.

5.7.1 Definition

fun *classic-nanson-rule* :: *'a Electoral-Module* **where**
  classic-nanson-rule A p =
    ((*leq-average-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) *A p*

fun *classic-nanson-rule-code* :: *'a Electoral-Module* **where**
  classic-nanson-rule-code A p =
    ((*leq-average-eliminator borda-score-code*) $\circlearrowleft_{\exists!d}$) *A p*

end

5.8 Schwartz Rule

theory *Schwartz-Rule*
  imports *Compositional-Structures/Basic-Modules/Borda-Module*
    Compositional-Structures/Defer-One-Loop-Composition
begin

This is the Schwartz voting rule. Confusingly, it is sometimes also referred as Nanson's rule. The Schwartz rule proceeds as in the classic Nanson's rule, but excludes alternatives with a Borda score that is strictly less than the average Borda score.

5.8.1 Definition

fun *schwartz-rule* :: *'a Electoral-Module* **where**
  schwartz-rule A p =
    ((*less-average-eliminator borda-score*) $\circlearrowleft_{\exists\,!d}$) *A p*

fun *schwartz-rule-code* :: *'a Electoral-Module* **where**
  schwartz-rule-code A p =
    ((*less-average-eliminator borda-score-code*) $\circlearrowleft_{\exists\,!d}$) *A p*

end

5.9 Sequential Majority Comparison

theory *Sequential-Majority-Comparison*
  imports *Compositional-Structures/Basic-Modules/Plurality-Module*
    Compositional-Structures/Drop-And-Pass-Compatibility
    Compositional-Structures/Revision-Composition
    Compositional-Structures/Maximum-Parallel-Composition
    Compositional-Structures/Defer-One-Loop-Composition
begin

Sequential majority comparison compares two alternatives by plurality voting. The loser gets rejected, and the winner is compared to the next alternative. This process is repeated until only a single alternative is left, which is then elected.

5.9.1 Definition

fun *smc* :: *'a Preference-Relation* ⇒ *'a Electoral-Module* **where**
  smc x A p =
    ((((((*pass-module 2 x*) ▷ ((*plurality*↓) ▷ (*pass-module 1 x*))) $\|_\uparrow$
    (*drop-module 2 x*)) $\circlearrowleft_{\exists\,!d}$) ▷ *elect-module*) *A p*)

5.9.2 Soundness

theorem *smc-sound*:
  assumes *order*: *linear-order x*
  shows *electoral-module* (*smc x*)
  unfolding *electoral-module-def*
proof (*simp*, *safe*, *simp-all*)
  fix
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    xa :: *'a*
  let *?a* = *max-aggregator*
  let *?t* = *defer-equal-condition*
  let *?smc* =
    pass-module 2 x ▷
      ((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
        drop-module 2 x ↺$_?$*t* (*Suc 0*)
  assume
    fin-A: *finite A* **and**
    prof-A: *profile A p* **and**
    reject-xa:
      xa ∈ *reject* (*?smc*) *A p* **and**
    elect-xa:
      xa ∈ *elect* (*?smc*) *A p*
  show *False*
    using *IntI drop-mod-sound elect-xa emptyE fin-A*
          loop-comp-sound max-agg-sound order prof-A
          par-comp-sound pass-mod-sound reject-xa
          plurality-sound result-disj rev-comp-sound
          seq-comp-sound
    by *metis*
next
  fix
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    xa :: *'a*
  let *?a* = *max-aggregator*
  let *?t* = *defer-equal-condition*
  let *?smc* =
    pass-module 2 x ▷
      ((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
        drop-module 2 x ↺$_?$*t* (*Suc 0*)
  assume
    fin-A: *finite A* **and**
    prof-A: *profile A p* **and**
    reject-xa:
      xa ∈ *reject* (*?smc*) *A p* **and**
    defer-xa:
      xa ∈ *defer* (*?smc*) *A p*
  show *False*

```
    using IntI drop-mod-sound defer-xa emptyE fin-A
          loop-comp-sound max-agg-sound order prof-A
          par-comp-sound pass-mod-sound reject-xa
          plurality-sound result-disj rev-comp-sound
          seq-comp-sound
    by metis
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    xa :: 'a
  let ?a = max-aggregator
  let ?t = defer-equal-condition
  let ?smc =
    pass-module 2 x ▹
       ((plurality↓) ▹ pass-module (Suc 0) x) ∥?a
         drop-module 2 x ↻?t (Suc 0)
  assume
    fin-A: finite A and
    prof-A: profile A p and
    elect-xa:
      xa ∈ elect (?smc) A p
  show xa ∈ A
    using drop-mod-sound elect-in-alts elect-xa fin-A
          in-mono loop-comp-sound max-agg-sound order
          par-comp-sound pass-mod-sound plurality-sound
          prof-A rev-comp-sound seq-comp-sound
    by metis
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    xa :: 'a
  let ?a = max-aggregator
  let ?t = defer-equal-condition
  let ?smc =
    pass-module 2 x ▹
       ((plurality↓) ▹ pass-module (Suc 0) x) ∥?a
         drop-module 2 x ↻?t (Suc 0)
  assume
    fin-A: finite A and
    prof-A: profile A p and
    defer-xa:
      xa ∈ defer (?smc) A p
  show xa ∈ A
    using drop-mod-sound defer-in-alts defer-xa fin-A
          in-mono loop-comp-sound max-agg-sound order
          par-comp-sound pass-mod-sound plurality-sound
          prof-A rev-comp-sound seq-comp-sound
```

by (*metis* (*no-types*, *lifting*))
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
xa :: *'a*
let *?a* = *max-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
drop-module 2 x ↺$_?$*t* (*Suc 0*)
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
reject-xa:
xa ∈ *reject* (*?smc*) *A p*
have *plurality-rev-sound*:
electoral-module
(*plurality*::*'a set* ⇒ (- × -) *set list* ⇒ - *set* × - *set* × - *set*↓)
by *simp*
have *par1-sound*:
electoral-module (*pass-module 2 x* ▷ ((*plurality*↓) ▷ *pass-module 1 x*))
using *order*
by *simp*
also have *par2-sound*:
electoral-module (*drop-module 2 x*)
using *order*
by *simp*
show *xa* ∈ *A*
using *reject-in-alts reject-xa fin-A in-mono*
loop-comp-sound max-agg-sound order
par-comp-sound pass-mod-sound prof-A
seq-comp-sound pass-mod-sound par1-sound
par2-sound plurality-rev-sound
by (*metis* (*no-types*))
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
xa :: *'a*
let *?a* = *max-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
drop-module 2 x ↺$_?$*t* (*Suc 0*)
assume
fin-A: *finite A* **and**

prof-A: *profile A p* **and**
xa-in-A: *xa* $\in$ *A* **and**
not-defer-xa:
xa $\notin$ *defer* (*?smc*) *A p* **and**
not-reject-xa:
xa $\notin$ *reject* (*?smc*) *A p*
show *xa* $\in$ *elect* (*?smc*) *A p*
using *drop-mod-sound loop-comp-sound max-agg-sound order par-comp-sound pass-mod-sound xa-in-A plurality-sound rev-comp-sound seq-comp-sound electoral-mod-defer-elem fin-A not-defer-xa not-reject-xa prof-A*
by *metis*
qed

5.9.3 Electing

theorem *smc-electing*:
assumes *order*: *linear-order x*
shows *electing* (*smc x*)
proof −
let *?pass2* = *pass-module 2 x*
let *?tie-breaker* = (*pass-module 1 x*)
let *?plurality-defer* = (*plurality*$\downarrow$) $\triangleright$ *?tie-breaker*
let *?compare-two* = *?pass2* $\triangleright$ *?plurality-defer*
let *?drop2* = *drop-module 2 x*
let *?eliminator* = *?compare-two* $\|_\uparrow$ *?drop2*
let *?loop* =
let t = *defer-equal-condition 1 in* (*?eliminator* $\circlearrowleft_t$)

have *00011*: *non-electing* (*plurality*$\downarrow$)
by *simp*
have *00012*: *non-electing ?tie-breaker*
using *order*
by *simp*
have *00013*: *defers 1 ?tie-breaker*
using *order pass-one-mod-def-one*
by *simp*
have *20000*: *non-blocking* (*plurality*$\downarrow$)
by *simp*

have *0020*: *disjoint-compatibility ?pass2 ?drop2*
using *order*
by *simp*
have *1000*: *non-electing ?pass2*
using *order*
by *simp*
have *1001*: *non-electing ?plurality-defer*
using *00011 00012*

```
    by simp
  have 2000: non-blocking ?pass2
    using order
    by simp
  have 2001: defers 1 ?plurality-defer
    using 20000 00011 00013 seq-comp-def-one
    by blast

  have 002: disjoint-compatibility ?compare-two ?drop2
    using order 0020
    by simp
  have 100: non-electing ?compare-two
    using 1000 1001
    by simp
  have 101: non-electing ?drop2
    using order
    by simp
  have 102: agg-conservative max-aggregator
    by simp
  have 200: defers 1 ?compare-two
    using 2000 1000 2001 seq-comp-def-one
    by auto
  have 201: rejects 2 ?drop2
    using order
    by simp

  have 10: non-electing ?eliminator
    using 100 101 102
    by simp
  have 20: eliminates 1 ?eliminator
    using 200 100 201 002 par-comp-elim-one
    by metis

  have 2: defers 1 ?loop
    using 10 20
    by simp
  have 3: electing elect-module
    by simp

  show ?thesis
    using 2 3 smc-sound smc.simps electing-def
          Defer-One-Loop-Composition.iter.simps
          order seq-comp-electing
    by metis
qed
```

5.9.4 (Weak) Monotonicity Property

theorem *smc-monotone*:

assumes *order*: *linear-order x*
shows *monotonicity* (*smc x*)
proof −

let *?pass2* = *pass-module 2 x*
let *?tie-breaker* = (*pass-module 1 x*)
let *?plurality-defer* = (*plurality*↓) ▷ *?tie-breaker*
let *?compare-two* = *?pass2* ▷ *?plurality-defer*
let *?drop2* = *drop-module 2 x*
let *?eliminator* = *?compare-two* $\|_\uparrow$ *?drop2*
let *?loop* =
let t = *defer-equal-condition 1 in* (*?eliminator* $\circlearrowleft_t$)

have *00010*: *defer-invariant-monotonicity* (*plurality*↓)
by *simp*
have *00011*: *non-electing* (*plurality*↓)
by *simp*
have *00012*: *non-electing ?tie-breaker*
using *order*
by *simp*
have *00013*: *defers 1 ?tie-breaker*
using *order pass-one-mod-def-one*
by *simp*
have *00014*: *defer-monotonicity ?tie-breaker*
using *order*
by *simp*
have *20000*: *non-blocking* (*plurality*↓)
by *simp*

have *0000*: *defer-lift-invariance ?pass2*
using *order*
by *simp*
have *0001*: *defer-lift-invariance ?plurality-defer*
using *00010 00011 00012 00013 00014*
by *simp*
have *0020*: *disjoint-compatibility ?pass2 ?drop2*
using *order*
by *simp*
have *1000*: *non-electing ?pass2*
using *order*
by *simp*
have *1001*: *non-electing ?plurality-defer*
using *00011 00012*
by *simp*
have *2000*: *non-blocking ?pass2*
using *order*
by *simp*
have *2001*: *defers 1 ?plurality-defer*
using *20000 00011 00013 seq-comp-def-one*

by *blast*

have *000*: *defer-lift-invariance ?compare-two*
using *0000 0001*
by *simp*
have *001*: *defer-lift-invariance ?drop2*
using *order*
by *simp*
have *002*: *disjoint-compatibility ?compare-two ?drop2*
using *order 0020*
by *simp*

have *100*: *non-electing ?compare-two*
using *1000 1001*
by *simp*
have *101*: *non-electing ?drop2*
using *order*
by *simp*
have *102*: *agg-conservative max-aggregator*
by *simp*
have *200*: *defers 1 ?compare-two*
using *2000 1000 2001 seq-comp-def-one*
by *auto*
have *201*: *rejects 2 ?drop2*
using *order*
by *simp*

have *00*: *defer-lift-invariance ?eliminator*
using *000 001 002 par-comp-def-lift-inv*
by *simp*
have *10*: *non-electing ?eliminator*
using *100 101 102*
by *simp*
have *20*: *eliminates 1 ?eliminator*
using *200 100 201 002 par-comp-elim-one*
by *simp*

have *0*: *defer-lift-invariance ?loop*
using *00*
by *simp*
have *1*: *non-electing ?loop*
using *10*
by *simp*
have *2*: *defers 1 ?loop*
using *10 20*
by *simp*
have *3*: *electing elect-module*
by *simp*

show *?thesis*
using *0 1 2 3*
Electoral-Module.monotonicity-def
Defer-One-Loop-Composition.iter.simps
smc-sound smc.simps order seq-comp-mono
by (*metis* (*full-types*))
qed

end

Bibliography

[1] K. Diekhoff, M. Kirsten, and J. Krämer. Formal property-oriented design of voting rules using composable modules. In S. Pekeč and K. Venable, editors, *6th International Conference on Algorithmic Decision Theory (ADT 2019)*, volume 11834 of *Lecture Notes in Artificial Intelligence*, pages 164–166. Springer, 2019.

[2] K. Diekhoff, M. Kirsten, and J. Krämer. Verified construction of fair voting rules. In M. Gabbrielli, editor, *29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2019), Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2020.